MASTER EN SCIENCES INFORMATIQUES

Développement d'applications intelligentes au-dessus d'un réseau de capteurs sans fil

Woine, Rémy

Award date:
2012

Awarding institution:
Universite de Namur

Facultés Universitaires Notre Dame de la Paix de Namur
Faculté d'informatique
Année académique 2011-2012

Développement d'applications intelligentes au-dessus d'un réseau de capteurs sans fil

Rémy Woine

Mémoire présenté en vue de l'obtention du grade de Master en Sciences Informatiques

"*In this context, the acceptation of the term* intelligence *is quite pragmatic, and basically relates to the human user's perception : a system is intelligent when it behaves in an intelligent way from the viewpoint of the observer/user, independently of the system's inner structure.*"

A. Omicini, G. A. Papadopoulos, 2001.

Résumé

A l'heure actuelle, l'impact énergétique d'un bâtiment dans son environnement suscite un vif intérêt du monde industrielle. Cet intérêt influence directement le développement d'applications ou de dispositifs permettant non seulement d'analyser, mais aussi de contrôler les facteurs clés qui engendrent des coûts indésirés non négligeables. La surveillance quotidienne, la vitesse de réaction ainsi que le contrôle dynamique des grandeurs physiques qui rythment les fluctuations énergétiques, forment un ensemble important de paramètres pour la mise sur pied de telles applications. Ce mémoire présente de manière non exhaustive, une liste des implémentations possibles pour le déploiement d'un réseau de capteur sans fil dans un bâtiment. On propose pour la gestion d'un tel système, le langage de coordination TuCSoN et la programmation des règles dynamiques à l'aide du langage ReSpecT. L'état de l'art de ces concepts débouche sur le développement d'une application vue comme un outil permettant la création d'un ensemble de règles régissant un environnement prédéfini. L'aspect bas niveau y est aussi implémenté, en présentant une vue plus proche du langage de coordination en lui-même. On conclura sur la portée de l'implémentation, illustrée par des exemples ciblés.

Mots clés : bâtiment, énergétique, capteur, réseau, sans fil, langage de coordination, TuCSoN, ReSpecT.

Abstract

At the present time, the energetic impact of a building in its environment is a real interest for the industrials. This interest involves directly the development of applications or devices which can not only analyze but also control the key factors that generate significant undesired costs. The daily monitoring, the reaction rate and the dynamic control of the physical variables that shape energetic fluctuations, are a set of main parameters for the development of such applications. This thesis presents a non-exhaustive list of possible implementations for deploying a wireless sensor network in a building. It is proposed to manage such a system, the coordination language TuCSoN and the dynamic rules programming using the ReSpecT language. The state of the art of these concepts leads to the development of an application as a tool for creating a set of rules governing a predefined environment. The low-level aspect is also implemented, by presenting a closer view of the coordination language itself. We conclude on the scope of the implementation, illustrated by targeted examples.

Kewords : building, energetic, sensor, network, wireless, coordination language, TuCSoN, ReSpecT.

Avant-propos

Ce mémoire n'est pas seulement l'aboutissement de mes études de master en informatique, mais aussi un défi personnel lancé il y a deux ans, résultant en une amélioration considérable de mes connaissances et une envie toujours croissante de découvertes.

Certes, il est évident que tout travail ne se réalise sans peine, car "*à vaincre sans péril, on triomphe sans gloire*". C'est la raison pour laquelle je me dois de remercier certaines personnes qui ont contribué, de près ou de loin, à l'achèvement de ce travail.

En haut de la liste, toujours présente, elle a su me soutenir et me supporter, me dire les mots adéquats aux bons moments et m'apporter le réconfort qui manquait parfois. Et ce, jusqu'au bout. Merci à toi Corynne.

Merci aussi à Lionel, pour ses relectures intensives, au pied levé.

Je terminerai en remerciant mon promoteur, Jean-Marie Jacquet, pour ses conseils bien avisés, ses explications toujours très claires et son temps, car lui seul sait combien il est précieux.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Le présent travail tentera d'expliquer, étapes par étapes, le déroulement du développement d'un système intelligent, appliquant un langage de coordination à la gestion des flux d'information amenés par un réseau de capteurs sans fil.

La première partie de ce travail commencera par faire un état de l'art de ce qui existe dans les technologies de capteurs sans fil. On tentera de dégager celles qui semblent les plus adéquates à remplir le rôle dont il est question ici, en comparant les divers avantages et inconvénients qu'une telle implémentation peut avoir.

S'en suit une description realtivement détaillée de l'espace de coordination de TuCSoN. On tentera d'aider le lecteur en exposant un maximum de prérequis, de manière à fournir une lecture plus fluide du document. La partie plus applicative relatant le langage de spécifications ReSpecT, jumelé avec l'environnement TuCSoN, montrera ce dont un système implémentant ce type d'architecture est capable de fournir.

Le développement d'une application telle que décrite dans le dernier chapitre rassemble les deux points précédents de façon à montrer une utilisation réelle de l'application d'un langage de coordination à un réseaux de senseurs.

On terminera par une conclusion globale sur le travail. De celle-ci découleront les aspects scolaires mais aussi personnels qu'un tel travail peut amener.

Chapitre 2

Contexte

Le contexte posé par ce travail s'oriente vers l'intégration dans un bâtiment, d'un système capable, sur base d'un ensemble prédéfini de règles, de décider d'une action à entreprendre. Ce système de règles est alimenté par un réseau de capteurs sans fil, répartis dans le bâtiment à des endroits judicieux, comme par exemple une salle de serveur, les cuisines ou encore les ascenceurs.

Les mesures sont effectuées sur des grandeurs physiques communes, à savoir la température, l'humidité ou la luminosité. Elles ont pour but d'effectuer une télémétrie pour un audit énergétique ou pour la surveillance d'évènements non désirés. Les conclusions tirées de l'analyse de ces valeurs, permettront de repenser l'organisation ou la gestion du bâtiment en lui-même.

L'application doit pouvoir être pilotée par des utilisateurs non familiers avec les systèmes d'informations en général. En effet, l'ensemble des règles, mais aussi la définition du bâtiment, devront être définis au préalable par une personne qui a coutume de travailler dans la bâtiment ou qui dispose de connaissances suffisantes pour définir le système en tant que tel. Ceci ayant pour but que le système comprenne correctement les tâches qu'il doit effectuer. Il est à noter que les règles, tout comme le bâtiment, peuvent évoluer au fil du temps. On doit donc tenir compte du changement dynamique du système.

Pour l'application intelligente gérant le système de règles et de décisions, le choix s'est porté sur un langage de coordination. Favorisant sensiblement l'échange d'information au travers d'une architecture distribuée, le langage de coordination TuCSoN utilisé permet aussi la gestion dynamique des espaces d'information alimentés par le réseau de capteurs sans fil.

Décrit de la sorte, le contexte de ce travail suppose donc le développement d'une application intelligente, dynamique et réactive, développée au-dessus d'un réseau de capteur sans fil dans un bâtiment.

Chapitre 3

Réseaux de capteurs sans fil

Les technologies sans fil, aussi nombreuses soient elles, constituent un réel attrait dans le monde actuel, en donnant tant au professionnel qu'au hobbyiste les possibilités d'imaginer sans cesse des applications de plus en plus innovantes.

Cependant, il est à remarquer qu'il persiste un certain manque de confiance dans le monde industriel quant au déploiement des technologies sans fil pour ce qui est lié, par exemple, au contrôle de processus ou à la logistique interne. Cette crainte est d'autant plus accrue lorsque le système met en jeu la sécurité, qu'elle touche les données (intégrité, piratage), le processus lui-même (dépassement de seuil, mauvais échantillonnage des données), ou le personnel de l'entreprise concernée (conséquences directes des points précédent).

Nous nous intéresserons ici, dans le cadre de ce travail, à un environnement plus fermé mettant un oeuvre le développement d'un réseau de capteurs sans fil dans un bâtiment, le tout ayant pour but la supervision de paramètres clés, tels la température, l'humidité et la luminosité.

Ce chapitre donnera une brève description d'un capteur dans son sens le plus général, pour ensuite s'orienter plus en détails vers les technologies de transmission sans fil.

3.1 Acquisition

Le capteur constitue l'élément de base du système complet en charge de l'acquisition et de la transmission de le grandeur physique mesurée. Qu'importe les différents types de mesures (température, humidité, luminosité) ou le procédé employé pour transmettre l'information, on peut schématiser le fonctionnement d'un capteur par le diagramme présenté ci-dessous :

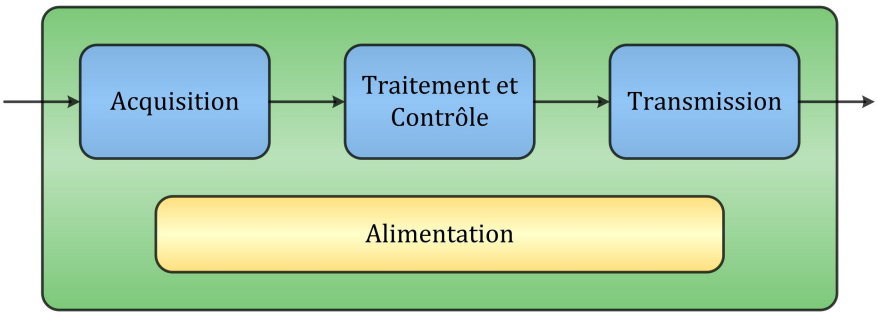

FIGURE 3.1 – Diagramme de fonctionnement d'un capteur

Chaque élément assure un rôle particulier dans un système d'acquisition de données.

Acquisition En première ligne dans la chaîne, ce bloc est propre à chaque type de mesure et permet la transformation directe de la grandeur physique à mesurer en une valeur électrique directement exploitable. Ce signal électrique est transformé par l'unité de conversion analogique/numérique en une valeur binaire qui sera envoyée au bloc suivant. C'est dans le premier

niveau d'acquisition que se posent les questions de résolution, de sensibilité, de précision ou encore de fidélité du capteur lui-même.

Traitement et Contrôle Ce bloc détient l'intelligence, plus ou moins élevée, du capteur. En effet, selon le type utilisé, on trouvera une gestion de l'envoi de l'information périodique ou événementielle (basée sur le dépassement de seuil), un système de gestion efficace de la batterie (avec mise en veille automatique), un système de cryptage des données voire même un système d'exploitation embarqué.

Transmission Ce dernier bloc va permettre d'envoyer les données sur un support de communication (médium). On trouve plusieurs variantes, aussi bien dans les modes de transmission filaire que non filaire. Le bloc de transmission renferme la pile de protocole qui s'occupera, par exemple, de la gestion des collisions ou du routage des données. Il est à noter que dans le cas des capteurs à intelligence élevée, les blocs de transmission et de contrôle se recouvrent.

Alimentation Ce bloc ne fait pas partie intégrante de la chaîne de fonctionnement du capteur comme les trois blocs précités, mais nécessite d'être énuméré pour attirer l'attention sur la problématique de l'alimentation électrique du capteur. En effet, cette problématique est surtout présente dans un système non filaire, où les principaux avantages sont l'absence de câbles et le caractère nomade de l'installation.

3.2 Transmission

Jusqu'à récemment, la seule manière de récolter des données entre un capteur et un client (point de collecte ou serveur) a été d'utiliser une transmission filiaire ou de collecter manuellement les données à l'aide d'un appareil dédié. Les technologies sans fil sont disponibles depuis des dizaines d'années. Cependant, elles ont pour défaut d'être relativement énergivores et de nécessiter des équipements spécialisés pour établir des communications. Grâce à l'introduction des puces électroniques dédiées aux technologies sans fil dites à faible puissance, de plus en plus de projets innovants commencent à voir le jour [13].

La plupart des marchés cibles sont caractérisés par un transfert périodique d'une petite quantité d'information entre un capteur et un composant central. Parmi les équipements implémentant une technologie de communication sans fil, on retrouve les téléphones portables, la domotique, les systèmes de chauffage, ventilation et climatisation, les commandes à distance, les jeux électroniques, les capteurs intelligents et bien d'autres.

Ces applications sont toutes contraintes par les exigences suivantes : faible puissance, faible coût, taille réduite. La première exigence relative à la puissance découle principalement des équipements fonctionnant pendant un temps prolongé à partir d'une simple batterie ("pile bouton"). Par exemple, supposons que le choix d'une puce électronique pour le déploiement de capteurs sans fil dans un bâtiment, a seulement su satisfaire les exigences de prix et de taille. Le coût de la maintenance pour le remplacement (périodique) de toutes les batteries prend largement le dessus sur les avantages d'une telle technologie. Il est donc nécessaire de tenir compte de l'entièreté du système afin de choisir la technologie sans fil adéquate.

FIGURE 3.2 – Pile bouton

3.2.1 Technologies sans fil

Les technologies sans fil utilisées actuellement peuvent majoritairement se regrouper en trois groupes distincts :

Infrarouge Les technologies infrarouges utilisent des rayons lumineux pour la transmission des données. Ces données sont échangées entre deux entités situées en vis-à-vis afin d'obtenir une ligne directe de visibilité entre l'émetteur et le récepteur. On retrouve ce type de transmission pour les télécommandes de téléviseur ou de porte de garage, par exemple.

Ondes radioélectriques Ce type de technologie utilise des ondes radioélectriques à basse ou haute fréquence pour l'échange d'information entre plusieurs dispositifs. C'est ce mode de transmission, et plus principalement celui utilisant les hautes fréquences, que l'on retrouve dans la plupart des systèmes actuels. Parmi les plus connus, la radio FM, le GSM, le WiFi, le Bluetooth et le ZigBee.

Champ proche Bien qu'utilisant aussi des ondes radioélectriques pour l'échange d'information, la technologie de communication en champ proche (en anglais, *NFC*[1]) peut être dissociée des ondes visées au point précédent au vu de son mode de fonctionnement relativement différent. On retrouve ce type de communication dans les techniques d'identification ou de contrôle d'accès utilisant la technologie RFID [2].

3.2.1.1 Propriétés

Suivant le projet dans lequel seront embarqués les dispositifs sans fil, le système va devoir faire face à des contraintes différentes ou évolutives. Avant de choisir, mais aussi d'utiliser au mieux, le type de protocole sans fil adéquat pour l'environnement prédéfini, il va falloir tenir compte de certaines propriétés telles que celles énumérées ci-après [13].

- ***Efficacité.*** Comme pour les communications filaires, l'efficacité du protocole de transmission peut être mesurée par le rapport entre la quantité de données utiles et la quantité de données réellement envoyées (en-tête + données utiles + code d'erreur). Cette efficacité est d'autant plus critique pour une communication sans fil. En effet, si un protocole a un très bas rendement et passe la plupart de son temps à transmettre des données non utiles, cela déchargera plus rapidement la batterie pour transférer, en réalité, très peu de données. A l'opposé, un protocole avec un rendement proche de l'unité transmettra beaucoup plus de données pour la durée de vie d'une même batterie. Il y a donc un compromis à faire entre l'efficacité et la fiabilité.
- ***Consommation.*** Une très basse consommation est souvent un paramètre clé pour les personnes désireuses de prolonger la vie des batteries de leurs équipements, amoindrissant de la sorte les coûts liés à la maintenance. Pour tout ce qui se rapporte aux capteurs sans fil, la notion de "*power per bit*" est à mettre en évidence pour le choix du protocole.
- ***Portée.*** La portée d'une technologie sans fil est souvent interprétée, à juste titre, comme étant proportionnelle à la sensibilité du récepteur et la puissance du transmetteur. Cependant, il existe bien d'autres paramètres pouvant affecter la portée réelle du dispositif sans fil. Parmi ceux-ci, l'environnement, la fréquence de l'onde porteuse, la qualité du design électronique et mécanique, les schémas de codage. Pour les applications reposant sur des capteurs sans fil, cette portée peut être un facteur important. De plus, dans la littérature (technique) la portée est souvent mentionnée pour un environnement idéal, mais les équipements sans fil sont souvent utilisés dans un environnement congestionné (au niveau de l'occupation du spectre de fréquence) et parsemé d'obstacles.

1. *Near Field Communication*
2. *Radio Frequency IDentification*

- ***Robustesse.*** Un transfert fiable d'un paquet de données a un impact direct sur la durée de vie de la batterie alimentant le dispositif sans fil. De manière générale, si un paquet n'a pas été délivré à cause de l'environnement, d'interférences ou de brouillage fréquenciel, le transmetteur réitérera l'envoi jusqu'à ce que le paquet soit délivré avec succès. Cela se produit évidemment au détriment de la durée de vie de la batterie. Si un système sans fil est restreint à un seul canal de fréquence pour sa communication, sa fiabilité peut se déteriorer dans des environnements congestionnés.

3.2.1.2 Topologies

Les topologies utilisées pour transmettre de l'information entre (au minimum) deux entités ont évolué au cours des années. Les premières à avoir été développées se voulaient simples mais robustes. A l'heure actuelle, on tend plus vers des topologies complexes, mais ayant la capacité de s'accroître facilement en gérant des plus en plus d'entités. Cela implique cependant une augmentation de la complexité du système mis en oeuvre.

- ***Point-to-Point.*** Communication point-à-point (figure 3.3). Seuls deux entités sont connectées sur une liaison directe. Le canal de communication ainsi créé, peut transporter l'information de manière unidirectionnelle dans un seul sens (*simplex)*, de manière bidirectionnelle mais en alternant les sens de communication (*half-duplex*) ou de manière bidirectionnelle où les deux entités peuvent échanger de l'information simultanément (*full-duplex*).

FIGURE 3.3 – Point-to-Point

- ***Broadcast.*** Mode émission (figure 3.4). Le dispositif envoie un message dans l'espoir qu'un récepteur le recevra à l'intérieur du rayon de portée. Le transmetteur, quant à lui, ne reçoit aucun signal. On retrouve ce type de topologie pour la transmission d'un signal de télévision. Dans certaines applications industrielles, un message *broadcast* spécifique peut être envoyé afin de synchroniser des équipements au sein d'une installation.

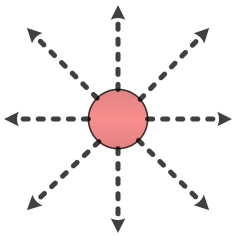

FIGURE 3.4 – Broadcast

- ***Scanning.*** Mode balayage (figure 3.5). Le dispositif est constamment en mode réception, attendant de recevoir un signal depuis n'importe quel appareil transmettant dans le rayon de portée. Les appareils déstinés à prendre des mesures dans un environnement afin d'en qualifier les perturbations utilisent une topologie balayage, comme par exemple pour les tests de compatibilité électromagnétique [3].

3. Laboratoire de compatibilité électromagnétique de l'Université de Liège : `http://ace.montefiore.ulg.ac.be/cem/`

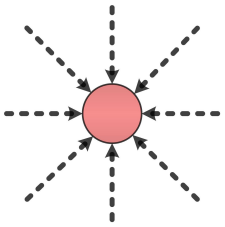

FIGURE 3.5 – Scanning

- ***Star.*** Réseau de communication en étoile (figure 3.6). Appelé aussi liaison maître-esclave, car seul le noeud central est en charge de l'établissement d'une communication avec les autres noeuds. Chaque noeud esclave a une liaison point-à-point avec le noeud central. On retrouve ce type de topologie dans une chaîne industrielle où l'intelligence est centralisée dans un seul dispositif (un automate) délèguant certaines tâches à des actionneurs (des contrôleurs de moteurs électriques, par exemple) ou rapatriant des informations de capteurs.

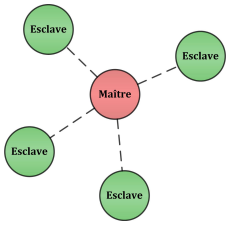

FIGURE 3.6 – Star

- ***Tree.*** Réseau de communication en arbre, appelé aussi liaison hiérarchique (figure 3.7). Chaque noeud a un noeud supérieur par lequel il est obligé de passer pour router l'information. Le noeud le plus haut dans la hiérarchie collecte alors les informations de tous les autres noeuds. L'inconvénient de ce type de réseau est que les noeuds de bas niveau sont tributaires du bon fonctionnement de leur supérieur hiérarchique.

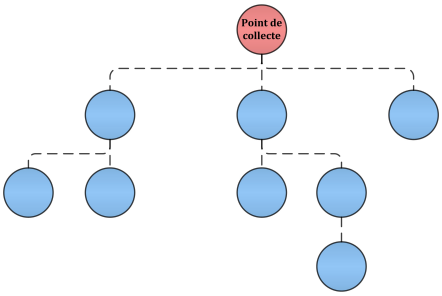

FIGURE 3.7 – Tree

- ***Mesh.*** Réseau de communication maillé (figure 3.8). Chaque noeud a une liaison point-à-point avec les noeuds voisins à portée. Cette topologie facilite le routage de l'information, le déploiement du système dans son environnement ainsi que l'ajout dynamique de noeuds dans le système. Le point de collecte d'information peut être placé n'importe où dans le réseau. Il est évident que ce type de topologie nécessite une gestion plus adéquate du système au vu du nombre de chemin que peut emprunter l'information. Plusieurs politiques de routages peuvent être adoptées suivant que l'on s'oriente vers un système de routage rapide ou un système de routage robuste (dualité des deux propriétés).

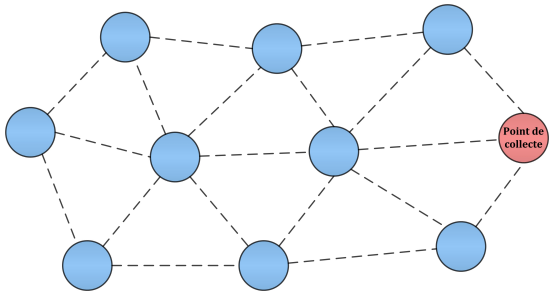

FIGURE 3.8 – Mesh

3.2.1.3 Collecte des données

Les données au sein d'un réseau de capteurs peuvent être rapatriées de manières différentes :
- **Cyclique.** Le capteur est "réveillé" à intervalle régulier pour envoyer l'information.
- **A la demande.** Un message de type *broadcast* est envoyé pour demander l'information.
- **Evénementielle.** L'envoi est déclenché par un évènement tel un dépassement de seuil.

Aussi, la manière dont les données vont être routées au travers d'un réseau de capteurs va dépendre des régles préétablies par le protocole de communication ou par la programmation du capteur lui-même. On trouvera de manière générale :
- **Routage *Flow-oriented*.** L'envoi de l'information est de type *broadcast*. Les capteurs qui recoivent le message le renvoient de la même manière (en excluant le destinataire de la liste de diffusion).
- **Routage pro-actif.** Le chemin que doit prendre l'information est régi par des tables de routage contenue dans le capteur. Ces tables doivent se mettre à jour régulièrement afin de garantir un routage toujours optimal.
- **Routage réactif.** Le chemin de routage est établi à la demande. Le capteur découvre lui-même l'environnement avoisinnant lors de l'envoi de l'information.
- **Routage hybride.** Plutôt dédié à un grand réseau de capteurs. Le routage hybride est un mélange des routages pro-actif et réactif où deux niveaux de routage sont utilisés, l'un pour une zone proche (*pro-actif*) et l'autre pour une zone lointaine (*réactif*).

3.2.1.4 Zone de couverture

La zone de couverture va définir le champ de perception d'un réseau de capteurs au sein de la zone d'intérêt à sonder. Le rapport entre la zone de couverture et la zone d'intérêt peut varier en fonction des spécifications du système d'acquisition, selon que l'on désire une couverture globale (mesures précises et cartographie complète de la zone d'intérêt) ou un seul point de référence (moyenne de la mesure sur la surface de la zone d'intérêt).

Pour qu'une zone soit complètement couverte, il faut non seulement que le champ de perception des capteurs qui la constituent se recouvrent (afin d'éviter les points aveugles) mais aussi que la distance entre chaque capteur permette toujours la communication pour le routage de l'information.

3.2.1.5 Exemples

Il est évident que beaucoup de protocoles ont vu le jour depuis la première transmission sans fil. L'engouement des ingénieurs et des spécialistes ont permis la création de méthodes de communication de plus en plus innovantes. Certaines méthodes sont restées attachées à un système plus fermé, tandis que d'autres se voulaient plus génériques afin de toucher un plus large marché.

- ***IrDA.*** Acronyme de *Infrared Data Association*. Même si l'arrivée de la version ultra haute vitesse (1 Gbps) peut sembler prometteuse, ce protocole se voit contraint par sa portée réduite de quelques dizaines de centimètres et par le fait qu'il faut garder une visée directe entre les deux éléments communicants. De plus, on notera que l'IrDA est parfois plus gourmand en énergie que les technologies par ondes radios. Cependant, il reste assez utilisé dans des environnements facilement perturbés par des interférences et sa facilité de mise en oeuvre est un atout non négligeable. Aussi, certains préfèreront utiliser ce type de protocole par sa nature "*green*", c'est-à-dire qu'il n'émet pas d'ondes à tout va dans l'environnement avoisinant [5].

FIGURE 3.9 – IrDA

- ***Bluetooth.*** Initialement développé pour éliminer le câblage entre périphériques, le Bluetooth est sensiblement dédié au transfert de la voix et de données multimédia, dans un environnement de style PAN [4]. Relativement gourmand en énergie pour être employé, par exemple, pour des transferts de données entre capteurs sans fil, ce type de communication est aussi difficile à mettre en oeuvre pour la mise en réseau de plusieurs éléments embarqués. L'arrivée de la version 4.0 du protocole permettra d'outrepasser ces contraintes en promettant un version allégée [5] et basse énergie, dédiée aux systèmes plus autonomes, alimentés sur une simple pile bouton [4].

FIGURE 3.10 – Bluetooth

- ***ANT.*** Protocole propriétaire spécialement développé par une division de *Dynastream Innovations* pour être implémenté dans un réseau de capteurs sans fil. Sa technologie se veut très basse consommation, peu chère et optimisée pour l'implémentation dans un design embarqué. Ce protocole a été initialement créé pour les appareils de sport et fitness en permettant à des capteurs de communiquer avec un petit écran pour une montre ou un vélo. La technologie ANT+ sous-jacente reprend le protocole ANT et rend les dispositifs intéropérables au sein d'un réseau "géré" (*managed network*), garantissant ainsi que tous les dispositifs marqués ANT+ travaillent de manière transparente [6].

FIGURE 3.11 – ANT+

4. *Portable Area Network*
5. La version allégée implémente seulement la nouvelle version du protocole, tandis que d'autres puces électroniques pourront aussi supporter toutes les versions antérieures.

- ***ZigBee.*** ZigBee est une spécification pour le sans fil basse consommation basée sur le standard IEEE 802.15.4.2003 [6]. Cette spécification a introduit les réseaux maillés et est généralement dédiée aux capteurs intelligents, à la domotique et aux contrôles à distance. Même si ZigBee semble avoir l'avantage pour les paramètres telles que la consommation et la facilité de mise en oeuvre en comparaison au Bluetooth ou au Wi-Fi, l'usage de ce type de protocole pour un système embarqué alimenté par batterie nécessite un monitoring régulier, surtout lorsqu'utilisé pour de longues périodes de temps [3].

FIGURE 3.12 – ZigBee

- ***RF4CE.*** Acronyme de *Radio Frequency for Consumer Electronics*, ce protocole est basé sur ZigBee [7]. L'utilité principale de cette spécification est de définir un standard pour un réseau de commandes à distance pour les appareils électroniques grand public, comme, par exemple, les décodeurs TV. La spécification se veut simple, de faible consommation et tend à résoudre les problèmes posés par l'utilisation du protocole IrDA (interopérabilité, visibilité directe et fonctionnalités limitées).

FIGURE 3.13 – RF4CE

- ***WiFi.*** Le standard de réseau sans fil WiFi (IEEE 802.11) n'a cessé de s'améliorer au fil des années, surtout au point de vue de la consommation d'énergie. Cependant, ce protocole est plus optimisé au transfert de grosse quantité de données et au haut débit, ce qui ne le rend pas vraiment adéquat pour un dispositif alimenté par batterie. De plus, l'implémentation d'un tel protocole pour un dispositif autonome est plus fastidieuse et semble être une solution bien trop lourde pour un système à capteur sans fil [2].

FIGURE 3.14 – WiFi

6. *Institue of Electrical and Electronics Engineers*
7. `http://www.zigbee.org/Specifications/ZigBeeRF4CE/Overview.aspx`

- ***NFC.*** Le protocole de communication en champ proche (*Near Field Communication*) est vraiment différent des protocoles précités utilisant les ondes radios. Ce type de communication a été créé pour établir un échange de données dans une portée limitée à quelques centimètres. Ce standard est basé sur les standards existant de RFID dédiés au contrôle d'accès et au petit transfert de données. Il est intéressant de citer qu'une communication peut exister entre un équipement NFC (maître) et un autre équipement NFC purement passif (esclave) ; l'énergie est alors apportée par l'onde porteuse qui véhicule l'information.

FIGURE 3.15 – NFC

On retrouve donc dans les exemples précités les trois grands types de communication sans fil. On remarque que le transfert de données par ondes radioélectriques est le plus usité à l'heure actuelle, et est aussi celui qui présente le plus de protocoles différents. Les technologies Bluetooth, ANT, ZigBee, RF4CE et WiFi utilisent toutes une onde porteuse à 2,4 GHz pour propager l'information, tandis que la communication en champ proche (NFC) travaille, de manière plus ponctuelle, avec une onde à 13,56 MHz. Les tableaux présentés ci-dessous permettent de se représenter plus facilement les caractéristiques générales de ces protocoles [13].

	IrDA	**Bluetooth**	**ANT**	**ZigBee**	**RF4CE**	**WiFi**	**NFC**
Point-toPoint	√	√	√	√	√	√	√
Broadcast		√	√				
Scanning		√	√	√	√		
Star		√	√	√	√	√	
Tree		√	√	√	√		
Mesh			√	√	√		

TABLE 3.1 – Topologies

	IrDA	**Bluetooth**	**ANT**	**ZigBee**	**RF4CE**	**WiFi**	**NFC**
Portée	10[cm]	280[m]	30[m]	100[m]	100[m]	150[m]	5[cm]
Débit	1[Gbps]	300[kbps]	20[kbps]	100[kbps]	100[kbps]	6[Mbps]	420[kbps]

TABLE 3.2 – Portée (environnement ouvert) - Débit (données utiles)

3.3 Conclusion

L'étude complète du système que l'on veut mettre en place ainsi que de ses exigences vont orienter fortement le choix du protocole sans fil. Certains marchés cibles ont déjà su imposer une technologie particulière, faisant alors accroitre sa réputation et permettre, de la sorte, une expansion un peu plus rapide (comme pour le Bluetooth et le WiFi, par exemple).

La technologie ANT a su s'intégrer dans les milieux du sports et du fitness grace à sa facilité de mise en oeuvre. Le fait de ne pas vouloir s'attaquer directement à de hauts débits sur des grandes distances a permis à cette technologie d'exploiter son côté facilement intégrable et basse consommation.

Bluetooth, et plus particulièrement sa version 4.0, se veut être un concurrent important pour ANT, ce qui permettra à cette technologie de toucher de plus vastes marchés, tout en gardant pour acquis son bagage sur les téléphones mobiles.

La technologie WiFi, initiallement créée pour le traffic de masse à haut débit, ne cesse de progresser dans ses versions basse consommation mais est tributaire de sa lourde pile de protocole, lui empèchant de s'intégrer dans des dispositifs plus nomades.

Les protocoles ZigBee et RF4CE sont virtuellement identiques. Ils arrivent en tête pour le déploiement d'un réseau de capteur sans fil grâce à la facilité d'exploitation d'un réseau maillé. Il reste cependant un effort à faire en terme de puissance consommée, en comparaison à la technologie ANT/ANT+.

Les communication en champ proche que couvre la technologie NFC n'est pas un concurrent direct pour le déploiement d'une technologie sans fil dans un système de capteurs intelligents. On retrouvera plutôt ce type de communication dans des systèmes plus adéquats aux applications dites "Touch to <action>" [13] comme le tracking, le contrôle d'accès et l'identification (humains et animaux).

Le faible coût de l'implémentation du protocole IrDA lui permet d'être toujours présent pour les commandes à (courtes) distances que l'on retrouve pour les téléviseurs ou les portes de garage. C'est une technologie utilisée depuis longtemps mais qui tend à être remplacée par un protocole de communication reposant sur les ondes radios au vue de la faible consommation que l'on peut (veut) obtenir à l'heure actuelle.

Pour le projet dont il est question dans ce rapport, le système le plus adéquat et le plus facile à mettre en oeuvre est basé sur le protocole ZigBee. Il existe en effet beaucoup plus de kits d'évaluation ou de petits boîtiers orientés développement que pour la technologie ANT/ANT+. Aussi, l'exercice étant de déployer des capteurs sans fil dans un bâtiment, un réseau maillé permet d'acheminer plus facilement l'information qu'une topolgie en étoile ou en arbre.

Chapitre 4

Application intelligente

L'intelligence (artificielle) du système qui régit l'application est basée sur un système de règles qui, à partir d'un ensemble de données provenant des grandeurs physiques mesurées et d'un ensemble de règles introduites par l'utilisateur du système, parvient à dégager des actions à entreprendre pour réagir à une série de faits reconnus comme pertinents. Un tel système peut être codé dans le langage de programmation logique **Prolog**.

Cependant, nous avons voulu nous tourner vers un système plus réactif, une sorte de "**Prolog** dynamique", permettant au système de s'auto-alimenter en introduisant d'autres règles suite aux différents évènements survenus. La base du système d'intelligence artificielle de notre application est développée autour de l'API **TuCSoN** (exécutée à partir du **Java**) dont le principal avantage est de donner aux processus constituant le système un comportement réactif programmable dynamiquement.

Les notions de **TuCSoN** ainsi que la majeure partie des explications décrites dans ce chapitre [1] (à l'exception des exemples et de l'introduction à **Prolog**) proviennent d'une traduction et d'une interprétation personnelle des informations issues du site Internet **aliCE Research Group** [1, 11]. Ce portail est maintenu par un groupe d'étudiants, de doctorants et de professeurs du *Department of Electronics, Informatics and Systems (DEIS)* ainsi que du *II Faculty of Computer Science and Engineering* de l'Université *Alma Mater Studiorum-Università* de Bologne (Italie) et vise à fournir une vision claire de leurs recherches. Il existe certains produits résultant de ces activités de recherche, dont **TuCSoN**, **tuProlog** et le langage **ReSpecT**.

4.1 Introduction à Prolog

Avant de démarrer les explications concernant **TuCSoN** et ses applications, certaines notions se doivent d'être explicitées afin de donner une lecture plus fluide des sections suivantes.

Le but premier de ces prérequis n'étant pas de relater un cours complet, mais bien d'éclaircir certaines notions qui seront considérées comme acquises lors du développement des sections ultérieures.

Prolog [2] est un langage de programmation logique fort utilisé dans les programmes d'intelligence articielle. Il permet de coder des systèmes experts génériques à chaînage arrière [3]. Il ne se présente pas sous la forme d'une séquence d'instructions destinées à être exécutées par l'ordinateur, comme la *programmation impérative*, mais sous la forme d'un certain nombre d'expressions logiques, exprimées en langage pseudo-naturel, formant une base de connaissances, représentant les affirmations connues sur le problème à traiter. Comme il s'agit de décrire le problème plutôt que d'expliciter des méthodes de résolution, on parle alors de *programmation déclarative*.

1. Il est à noter que la structuration des informations y est, toutefois, personnelle.
2. PROgrammation LOGique.
3. Le terme chaînage arrière exprime la manière dont le moteur d'inférence va traiter les faits pour la déduction et la résolution des requêtes.

Cette base de connaissances est constituée d'une base de *faits* et d'une base de *règles*. L'utilisation d'un programme se fait alors en posant des requêtes sur cette base de connaissances. **Prolog** dispose d'un *moteur d'inférence* qui lui permet de déduire certaines connaissances à partir des règles et des faits déjà présents dans sa base de connaissances.

4.1.1 Base de faits

La base de faits est donc composée d'affirmations. Chaque affirmation consiste en un *atome* contenant un ou plusieurs *termes* constants.

```
chat(tom).
pere(paul, marie).
```

Le premier prédicat signifie que *tom* est un *chat*. Le second indique que *paul* est le *père* de *marie*, on parle dans ce cas d'arité 2 au vu du nombre de termes. Ces éléments textuels doivent être une séquence de lettres, nombres ou sous-tirets commençant par une lettre minuscule.

4.1.2 Base de règles

La base de règles est composée de relations permettant de déduire des nouveaux faits. Chaque relation est composée d'un *antécédent* est d'une *conséquence*, eux mêmes composés d'*atomes* contenant un ou plusieurs *termes* (constants, variables ou structurés). Un terme constant est une séquence de lettres, nombres ou sous-tirets commençant par une lettre minuscule. Dans la règle ci-dessous, le terme `croquettes` est constant.

```
chat(X) :- mange(croquettes).
```

Une *variable* est une séquence de lettres, nombres ou sous-tirets commençant par une lettre majuscule.

```
chat(X) :- felin(X), petit(X).
pere(X, Y) :- homme(X), parent(X, Y).
```

Dans le premier exemple, on dira que 'X' est un chat si 'X' est un félin et que 'X' est petit. Dans le second exemple, le moteur d'inférence déduira que 'X' est le père de 'Y' si 'X' est un homme et que 'X' est un parent de 'Y'.

Il existe aussi des *variables anonymes*, notées '_'. On définit une *variable anonyme* comme une variable n'ayant pas d'utilité dans le processus de déduction de la règle en cours. Par exemple, dans la règle suivante :

```
pere(X) :- pere(X, Y).
```

où on définit que 'X' est père tout court, si 'X' est le père de quelqu'un (ici, 'Y'), on remarque que la variable 'Y' n'a pas d'utilité dans la déduction. On pourra alors écrire la règle avec une variable anonyme.

```
pere(X) :- pere(X, _).
```

Un terme structuré (ou complexe) est une fonction n-aire composé de 'n' termes. Par exemple, le prédicat

```
livre(dupont, dunod, 192-125678-67)
```

représente un livre dont l'auteur est `dupont`, l'éditeur `dunod` et l'isbn `192-125678-67`.

4.1.3 Requêtes

On exécute des requêtes sur la base de connaissances de manière à déduire de nouveaux faits, et de ce fait, résoudre un problème posé. Ainsi, la requête :

```
?- chat(tom).
```

va retourner la valeur '`true`'[4] car il existe bien un prédicat identique dans la base de connaissances. On peut aussi poser la question suivante :

```
?- chat(X).
```

Le programme **Prolog** va unifier la variable 'X' avec la ou les valeurs qu'il aura trouvées dans la base de connaissances, c'est-à-dire, tenter d'égaler les arguments deux à deux. En l'occurence, on obtiendra : `X = tom`.

Prolog peut travailler aussi de manière transparente aussi bien avec des nombres entiers que des nombres à virgule flottante. On retrouve dans le tableau 4.1 une série d'opérateurs arithmétiques et logiques utilisés par le programme **Prolog**.

+(X)	Plus X
-X	Moins X
X+Y	Addition
X-Y	Soustraction
X*Y	Multiplication
X/Y	Division (flottant)
X//Y	Division (entière)
X mod Y	Modulo (reste de la division entière)
float(X)	Entier positif le plus proche de X
X/\Y	ET bit-à-bit
X\/Y	OU bit-à-bit
X#Y	OU EXCLUSIF bit-à-bit
\(X)	Complément à 1 (inversion des bits)
X<<Y	Décalage de Y bits de X vers la gauche
X>>Y	Décalage de Y bits de X vers la droite
X=Y	Unification
X==Y	Egal
X\=Y	Différent
X<Y	Inférieur
X>Y	Supérieur
X=<Y	Inférieur ou égal
X>=Y	Supérieur ou égal

TABLE 4.1 – Opérateurs Prolog

Les concepts fondamentaux de la logique de déduction, c'est-à-dire du moteur d'inférence de **Prolog**, sont l'unification, la récursivité et le *backtracking* (retour sur trace, explicité plus loin dans le document). Ce moteur d'inférence est dit à chaînage arrière. Il cherchera dans les règles d'inférences de la base de connaissances celles dont la conséquence correspond au but recherché. Si l'antécédent de cette règle est elle-même composée de prédicats à vérifier, alors le moteur d'inférence devra trouver d'autres règles dont les conséquences lui permettent de satisaire tous ces buts.

Le chaînage avant se veut plus simple dans la recherche car il commence par alimenter sa base de connaissances avec tous les faits qu'il peut déduire des règles existantes, enrichissant de la sorte sa base de faits jusqu'à trouver la solution de la requête posée.

4. On suppose ici que l'on interroge un invite de commande **Prolog** qui aura été alimenté avec une base de connaissances.

4.1.3.1 Exemple de backtracking

Le *backtracking*, ou retour sur trace, consiste à revenir en arrière dans le chemin de décision pour sortir d'un blocage. On qualifie couramment la méthode des essais et erreurs comme une simple implémentation du *backtracking*.

Afin d'illustrer ce concept, on va considérer le problème suivant : un étudiant doit parcourir un labyrinthe pour aller chercher son diplôme (figure 4.1). La position de départ est donnée par les coordonnées `(1,1)`. L'étudiant ne peut se déplacer que d'un pas horizontal (axe des abscisses) ou que d'un pas vertical (axe des ordonnées) à la fois.

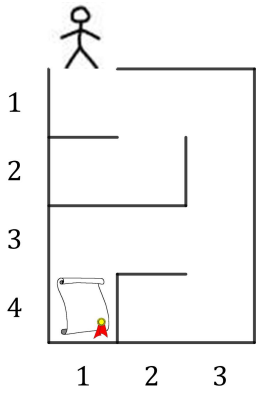

FIGURE 4.1 – Labyrinthe

L'arbre complet de décisions est donné par la figure 4.2. Arrivé à l'étape '**B**', deux choix s'offrent à l'étudiant. En choisissant la branche constituée des étapes '**C**' et '**D**', il arrive à un blocage. C'est donc ici que s'applique le *backtracking*, l'étudiant fait marche arrière pour remonter à l'étape '**B**' afin de choisir un autre chemin (l'étape '**C**' n'étant pas une étape constituée de choix). Arrivé à l'étape '**G**', si l'étudiant choisi la branche constitué des étapes '**H**' et '**I**', il sera bloqué. Le *backtracking* le ramènera à l'étape '**G**' où il poursuivra son avancée jusqu'à atteindre l'étape '**L**', étape finale où se trouve le but de la recherche (ici, le diplôme).

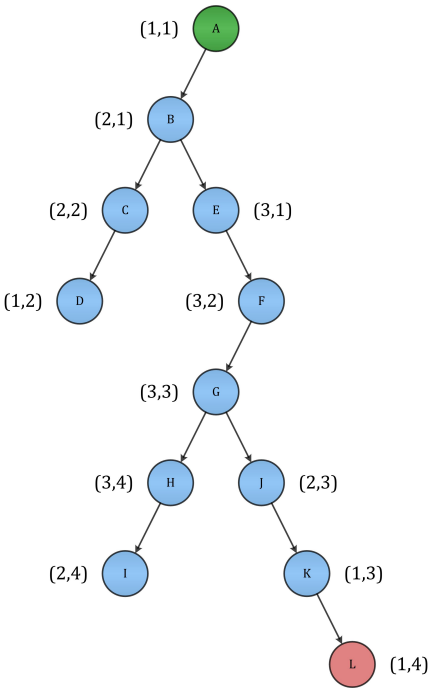

FIGURE 4.2 – Arbre de décisions

4.1.4 Prolog et TuCSoN

On retrouvera dans l'implémentation de **TuCSoN**, des prédicats du premier ordre semblables à ceux utilisés dans **Prolog**. On appelle prédicat du premier ordre, un atome qui n'accepte que des termes individuels tels ceux présentés un peu plus haut. Il existe des prédicats d'ordre supérieur prenant alors en argument, un ou plusieurs prédicats, ou encore, une ou plusieurs expressions. Un exemple de tel prédicat est :

```
predicat2(predicat1(X)).
```

4.2 Introduction à TuCSoN

TuCSoN (*Tuple Centre Spread over the Network*) est une infrastructure fournissant des services permettant la communication et la coordination de processus [5] distribués/concurrents appelés *agents*. Dans ce contexte, n'importe quel processus qui intégre sa propre structure de contrôle peut être considéré comme un *agent*.

Comme les objets dans la *Programmation Orientée Objet*, ces *agents* englobent un état et un comportement. Ils ont, par contre, le contrôle des deux. En effet, les objets de la POO n'ont pas le contrôle de leur comportement : envoyer un message à un objet implique l'invocation et l'exécution d'une méthode. Les *agents* n'interagissent donc pas au moyen d'invocation de méthodes mais plutôt au travers d'un modèle d'interaction ou de coordination, basé sur l'envoi de message et une communication partagée (tels les *blackboards* ou les *tuple spaces* [6]). L'infrastructure **TuCSoN** supporte la communication et la coordination d'*agents* fournissant des *tuple centres*, qui sont des espaces partagés et réactifs d'information, distribués au-dessus d'une infrastructure de "noeuds", où les *agents* insèrent, prélèvent, lisent de l'information sous forme de *tuples*, collections hétérogènes ordonnées de pièces d'information. Plus spécifiquement, les *tuple centres* sont des *tuple spaces* programmables, c'est-à-dire des *tuple spaces* avec un comportement réactif qui peut être programmé dynamiquement (par les humains ou par les *agents* eux-mêmes). Les *agents* accèdent aux *tuple centres* en écrivant, lisant ou consommant des *tuples* via de simples opérations de communication (ces opérations seront définies plus loin dans le document).

A l'origine, le modèle *tuple space* a été introduit comme un moyen puissant pour assurer la coordination dans les systèmes concurrents, en alternative aux modèles basés sur les messages ou sur les variables partagées. Parmi les points clés qui ont rendu les modèles *tuple space* différents des modèles pré-cités, on retrouve le style de communication générative : les *agents* communiquent respectivement en créant et en prélevant des *tuples* dont l'existence, une fois créée, est indépendante de celle des *agents*. Cela a rendu possible l'obtention de propriétés (spatiales et temporelles) qui simplifient grandement le développement de systèmes dans l'espace d'interaction des *agents*. Il est à noter que la communication générative est aussi liée à l'idée d'accéder à l'information en la spécifiant et non en accédant à une mémoire d'adresse donnée.

Les *tuple centres* spécialisent et étendent le modèle basique de *tuple space* en utilisant des *logic tuples* : les *tuples* sont des termes de logique du premier ordre (tels les termes en **Prolog**). Aussi, alors que le comportement d'un *tuple space* est figé en réponse aux évènements de communication, le comportement d'un *tuple centre* peut être ajusté par les besoins de l'application en définissant un ensemble de *specification tuples* exprimés dans le langage **ReSpecT**, qui définit comment un *tuple centre* devrait réagir aux évènements de communication entrante et sortante. De plus, contrairement aux *tuple spaces*, les *tuple centres* peuvent être programmés avec des réactions dans le but d'encapsuler des lois de coordination directement dans le medium de coordination.

5. On définit ici un processus par un composant software indépendant.
6. Les notions de *blackboards* ou *tuple spaces* seront détaillées plus loin dans le document.

Afin d'exploiter les services de **TuCSoN**, on utilisera l'API[7] (**Java**) dans sa dernière version stable (1.4.5). Cette API fournit les packages principaux nécessaires à la programmation d'*agents* interagissant avec des *tuple centres* : `alice.tucson.api` et `alice.tucson.logictuple`. Afin d'exploiter les services de **TuCSoN**, l'API fournit aussi un *Command Line Interpreter* et un *Inspector Tool*. Ces deux outils permettront d'inspecter et de débugger les *tuple centres* lors du développement ou de l'exploitation du système.

FIGURE 4.3 – Logo TuCSoN

4.3 Infrastructure de coordination

4.3.1 Modèle de coordination

Le terme "coordination" signifie fondamentalement la gestion de l'interaction des entités d'un système, qu'il s'agisse d'*agents*, de processus, de molécules, d'individus ou de tout autre élément. D'une manière générale, la notion de coordination est multidisciplinaire, et a donné plusieurs définitions spécifiques dans les nombreux domaines de recherche où elle est pertinente et couramment utilisée, tels que les langages de programmation, les systèmes parallèles et distribués, les systèmes (distribués) d'intelligence artificielle ou *multi-agents*, les technologies relatives à Internet, ainsi que le génie logiciel.

D'un point de vue commun en intelligence artificielle (distribuée), on peut voir la coordination comme le résultat de l'attitude de chaque individu au travers de l'organisation ou de la société à laquelle il appartient. Les croyances, les buts et les rôles de chaque individu, ainsi que sa perception subjective des interdépendances entre les membres d'une société, déterminent la coordination du système global. Même d'un point de vue général de la coordination dans les sociétés d'*agents* artificiels, les problèmes individuels et globaux de la coordination sont mélangés de telle manière qu'on ne puisse les distinguer facilement.

Cependant, la séparation des intérêts est généralement le moyen le plus facile de réduire la complexité. En effet, charger un *agent* de ses objectifs individuels et du poids de gérer toutes les subtilités de l'interaction dans des environnements qui peuvent être ouverts, physiquement distribués, et imprévisibles, ne rend pas facile les tâches de conception et de développement d'*agent*. Au lieu de cela, un modèle de coordination gardant la perception individuelle de la coordination distincte des problèmes de coordination globale rendrait possible la modélisation et la concrétisation de l'espace d'interaction et, en quelque sorte, indépendante des entités interagissantes. C'est ce qui a été appelé la *coordination objective*, car elle fait abstraction de la vue subjective de la coordination, ou aussi *coordination découplée*, car la coordination n'est plus couplée avec les problèmes d'exécution ou de "computation" (*computational issues*) des entités coordonnées.

La séparation entre exécution (computation) et coordination a été initiallement introduite dans le cadre des langages dédiés aux systèmes parallèles et distribués, et a montré clairement la distinction de ces deux concepts comme des dimensions orthogonales. Cela suggère que des modèles de coordination appropriés, approuvant les adjectifs précités, à savoir une coordination *objective* et *découplée*, peuvent avoir un impact significatif sur l'ingénierie des systèmes complexes, dont, évidemment, les applications intelligentes.

7. *Application Programming Interface.*

C'est précisément la notion spécifique de *modèle de coordination* telle qu'elle se manifeste dans les recherches sur les modèles et les langages de coordination. Ici, un modèle de coordination est, avant tout, une structure pour modéliser un espace d'interaction. Plus précisément, on peut imaginer un modèle de coordination comme la somme de trois éléments :

- *Les coordinables.* Les entités dont l'interaction mutuelle est régie par le modèle.
- *Les media de coordination.* Les abstractions permettant l'interaction entre les *coordinables* (par exemple, les sémaphores, les canaux, les *blackboards*, ...).
- *Les lois de coordination.* Les règles gouvernant l'interaction entre les *coordinables* et le média de coordination, aussi bien que le comportement d'un média de coordination en réponse aux événements d'interaction.

Les média de coordination, comme les *tuple spaces*, peuvent être exploités comme le noyau autour duquel le composant d'un système intelligent peut être organisé. Plus précisément, en travaillant comme un endroit naturel où les lois de coordination peuvent être incorporées, un médium de coordination permet au concepteur d'un système intelligent de définir les règles de coordination séparément des entités interagissantes, et de les encapsuler dans une abstraction dédiée.

Cela suggère comment les modèles et les langages de coordination peuvent influer la conception et le développement de systèmes intelligents. Cela fournit notamment aux ingénieurs de systèmes intelligents les propriétés fondamentales comme la *modularité* et la *réutilisabilité*. Un composant encapsulant une certaine forme d'intelligence, comme un système expert ou un *agent* intelligent, peut en fait fonctionner comme un "module intelligent" et être réutilisé partout où ses capacités sont nécessaires, en définissant les règles de coordination selon le modèle d'interaction du composant, et en les encapsulant dans un médium de coordination. Aussi, puisqu'ils sont encapsulés dans différents composants, les capacités individuelles et les règles de coordination peuvent être affinées et modifiées indépendamment, offrant ainsi des possibilités de conception et développement incrémantales.

D'un point de vue de l'ingénierie, un modèle de coordination peut être exploité comme un point de départ pour les métaphores, les abstractions et les mécanismes de conception qui sont nécessaires pour appuyer la définition de l'architecture d'un système intelligent, aussi bien que son développement et son déploiement.

L'origine de la littérature sur les modèles et les langages de coordination remonte à **Linda**, qui est peut-être le seul modèle de coordination connu en dehors des limites du domaine de la recherche. Bien que **Linda** ait été conçu dans le domaine de la programmation parallèle, avec des implémentations fermées et des techniques d'optimisation basées sur le compilateur, ses caractéristiques le rendent aussi viable pour des systèmes ouverts, et ont menés à la définition d'une multitude de dérivés des modèles *tuple-based* pour la coordination de systèmes distribués multi-composant ouverts et hétérogènes. En particulier, plusieurs modèles de coordination basés sur les *tuples* ont été définis pour des systèmes *multi-agents* et ont aussi été utilisés pour la coordination d'*agents* ou de composants intelligents.

Cependant, avant qu'ils soient effectivement exploités dans l'ingénierie des systèmes intelligents, les modèles de coordination doivent répondre à certaines questions clés comme la sécurité. L'authentification et l'autorisation sont des problèmes pertinents lorsqu'on considère que les systèmes intelligents tendent à gagner de plus en plus d'acceptation, devenant une technologie de tous les jours sur laquelle on peut se fier et agissant en tant que telle [10].

4.3.2 L'espace de coordination de TuCSoN

L'espace de coordination de **TuCSoN** appartient à une organisation. Cet espace est un ensemble de "noeuds" distribués au travers d'un réseau, où chaque noeud est identifié au moyen d'une adresse IP. Chaque noeud peut contenir un nombre illimité de *tuple centres* pour autant qu'ils aient un nom unique. Les *agents* présents dans cette organisation peuvent accéder de manière transparente soit aux *tuple centres* locaux soit aux *tuple centres* contenus dans un noeud distant.

Par facilité d'utilisation dans un espace ouvert, **TuCSoN** n'exige pas la création explicite du *tuple centre*, c'est-à-dire que l'infrastructure locale crée automatiquement le *tuple centre* lorsqu'il est référencé dans une opération.

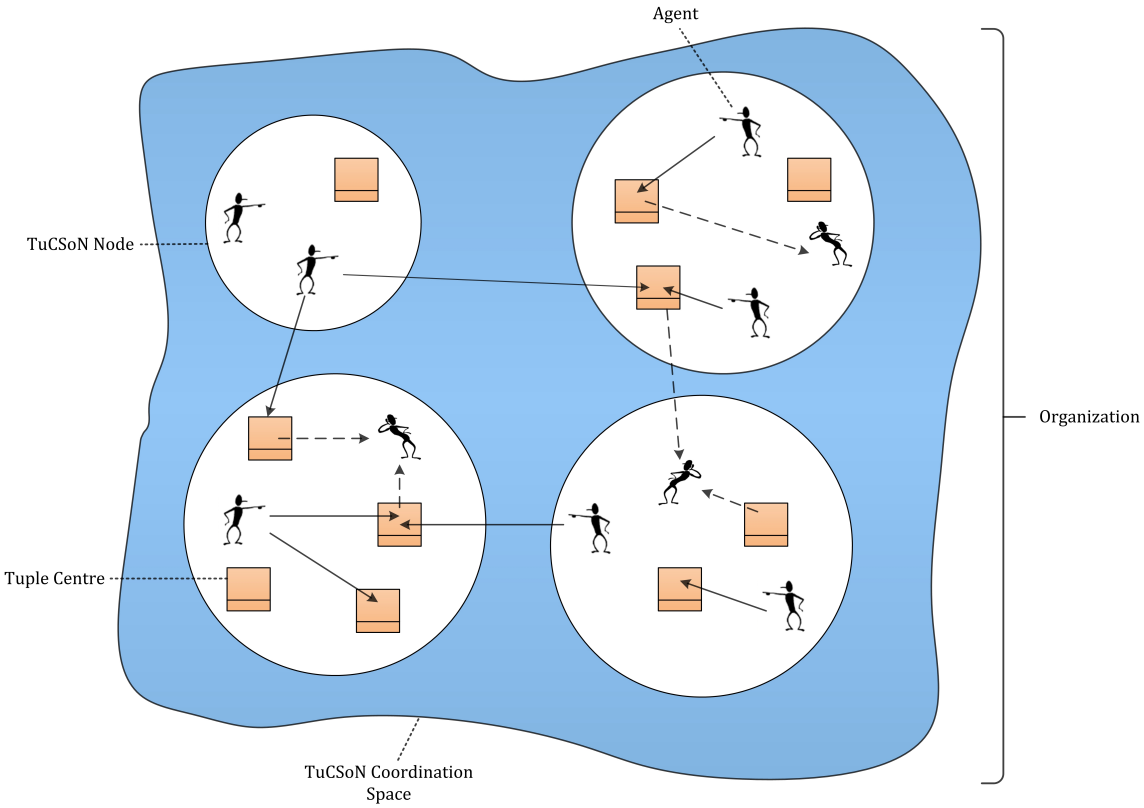

FIGURE 4.4 – Espace de coordination TuCSoN

Les *agents* accèdent aux noeuds grâce à des opérations basiques de communication telles que listées dans le tableau 4.2.

out	insère le *tuple* spécifié dans le *tuple centre* spéficié
in	retire un *tuple* qui correspond au *tuple template* spécifé du *tuple centre* spécifié (opération bloquante)
inp	retire un *tuple* qui correspond au *tuple template* spécifé du *tuple centre* spécifié (opération non bloquante ; échoue si aucun *tuple* ne correspond)
rd	lit un *tuple* qui correspond au *tuple template* spécifé du *tuple centre* spécifié (opération bloquante)
rdp	lit un *tuple* qui correspond au *tuple template* spécifé du *tuple centre* spécifié (opération non bloquante ; échoue si aucun *tuple* ne correspond)
set_spec	établit le comportement du *tuple centre* spécifié ; le tuple doit être écrit en langage **ReSpecT**
get_spec	renvoie le comportement du *tuple centre* spécifié

TABLE 4.2 – Opérations de communication

L'outil **TuCSoN** CLI fourni avec l'API permet l'invocation de commandes du langage de coordination **TuCSoN** sous la forme :

TCName @ TCAddress ? op(Tuple)

où

TCName est le nom du *tuple centre* : n'importe quelle chaîne de caractères peut être utilisée. Dans le cas de chaîne de caractères avec des espaces, la chaîne doit être entourée d'apostrophes.

TCAddress est l'adresse du *tuple centre* définie par un nom d'hôte ou une adresse IP.

op est l'opération à être exécutée parmi celles listées dans le tableau 4.2.

Tuple est l'information impliquée dans l'opération. Elle doit être sous la forme d'un *logic tuple* ou d'un *tuple template*, c'est-à-dire un terme **Prolog**.

Une caractéristique clé dans le modèle **TuCSoN** est de fournir une double vue de l'espace de coordination, qui peut être vu aussi bien comme une ressource globale que comme une ressource locale. Telle une ressource globale, un *tuple centre* est disponible depuis n'importe où dans le réseau, en spécifiant explicitemment son nom et son adresse. Telle une ressource locale, un *tuple centre* est simplement référencé par son nom, étant donné que son adresse implicite est celle du noeud local. Dans ce cas, les commandes se simplifient sous la forme :

TCName ? op(Tuple)

De plus, dans le but de fournir un support facile à utiliser pour des besoins simples de coordination, **TuCSoN** définit un *default tuple centre* pour chaque noeud, qui est appelé `default` : si le nom du *tuple centre* est manquant, le *default tuple centre* est visé :

op(Tuple)

4.3.2.1 Exemple d'utilisation

L'infrastructure **TuCSoN** est établie en installant (au minimum) un noeud. Un noeud peut être démarré en lançant le service TuCSoN associé (via l'invite de commande, par exemple) :

```
java alice.tucson.service.Node
```

On peut également spécifier le répertoire (*dir*) de la librairie compilée de **TuCSoN** (nommée dans l'exemple ci-dessous `tucson.jar`).

```
java -cp dir/tucson.jar alice.tucson.service.Node
```

Le port par défaut est le 20504 mais un autre port peut être spécifié en utilisant l'argument `-port`.

```
java alice.tucson.service.Node -port 20600
```

Afin de vérifier si le service est actif, on peut invoquer la commande **telnet** sur l'adresse IP du noeud qui renverra les informations suivantes.

```
TuCSoN Infrastructure (version 1.4.5)
Current date XXXX
Running since YYYY
```

A partir de ce point, l'utilisateur est prêt à interagir avec l'infrastructure en insérant ou en enlevant des *tuples* des *tuple centres* du noeud en question, voire même en changeant leurs comportements.

En utilisant l'outil **TuCSoN** CLI, on parvient aisément à tester les primitives de communication. Cet outil est lancé de la même manière que pour le noeud. Le CLI est vu comme un *agent* normal dans l'infrastructure (figure 4.5).

```
java alice.tucson.tools.CLIAgent
```

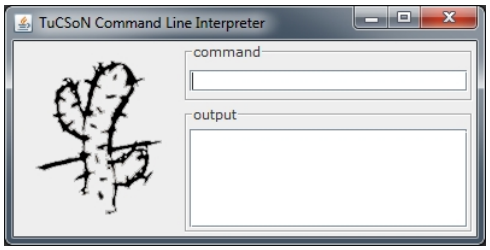

FIGURE 4.5 – Ecran d'accueil du CLI

A ce moment, le noeud est "vide", c'est-à-dire qu'il n'y a pas de *tuple centres*, donc pas de *tuples* non plus. On va insérer (figure 4.6) un premier *tuple* grâce à l'opération `out`. Vu qu'on ne spécifie pas le nom du *tuple centre*, le *tuple* sera inséré dans le *default tuple centre*.

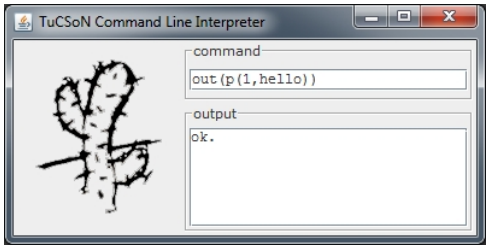

FIGURE 4.6 – Insertion d'un tuple

On peut vérifier l'existence de ce *tuple* avec l'opération `rd` ayant comme argument un *tuple template* de la forme `p(X,Y)`, c'est-à-dire un *tuple* nommé "`p`" avec deux variables [8]. L'*agent* CLI renvoie le premier *tuple* qu'il a trouvé (dans le *default tuple centre*) correspondant au *tuple template* spécifié (figure 4.7).

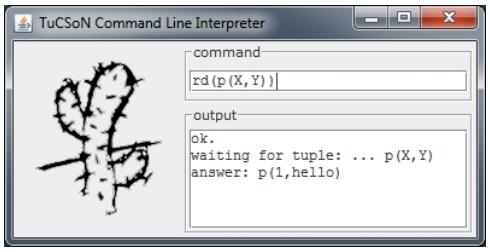

FIGURE 4.7 – Lecture d'un tuple

La figure 4.8 montre l'enlèvement du *tuple* correspondant au *tuple template* précédemment utilisé lors de l'opération `rd`. Afin de vérifier que le *tuple* est bien enlevé, on envoie une nouvelle commande de lecture. On utilisera la commande non bloquante `rdp` pour ne pas que le CLI attende en boucle (figure 4.9).

On remarque que la réponse affichée est bien `no`. Si on avait utilisé l'opération bloquante, aussi bien pour la commande `rd` que la commande `in`, le CLI aurait attendu en boucle l'insertion d'un nouveau *tuple* correspondant au *tuple template* `p(X,Y)`.

8. Les variables commencent donc par une majuscule (comme en **Prolog**).

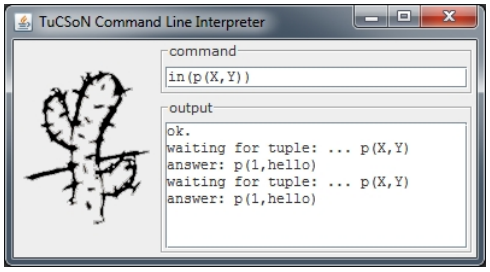

FIGURE 4.8 – Enlèvement d'un tuple

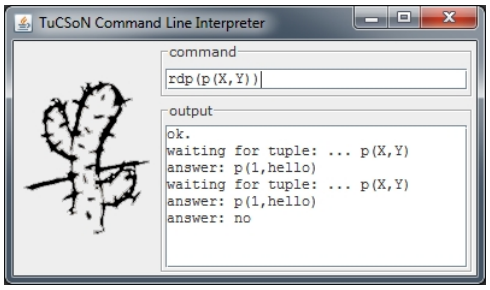

FIGURE 4.9 – Lecture non bloquante

4.4 Programmabilité dynamique

4.4.1 Des Tuple Spaces aux Tuple Centres

Les principaux avantages des modèles *tuple-based* sont la nette séparation entre exécution et coordination, la communication générative, et l'accès associatif à l'espace d'interaction. La première caractéristique prône une conception d'*agent* plus propre, en gardant l'exécution et la coordination clairement distinctes. La communication générative rend possible de découpler les *agents* aussi bien spatialement que temporellement, ce qui est presque obligatoire dans un environnement imprivisible tel que l'Internet, et plus particulièrement quand des *agents* mobiles sont concernés. L'accès associatif à l'information de communication rend les modèles *tuple-based* adéquats à traiter les systèmes hétérogènes et basés sur de l'information dynamique, toujours en faisant référence à l'Internet.

Un des grands atouts de l'intéraction au travers d'un *tuple space* est que la coordination est "dirigée par l'information" : les *agents* synchronisent, coopèrent et rivalisent, le tout étant basé sur l'information disponible dans un espace partagé de données, en accédant, consommant et produisant (associativement) de l'information. Alors que cela rend les protocoles d'intéraction simples mais expressifs, la représentation de l'information sous forme de *tuples* est aussi un des problèmes principaux de la coordination basée sur les *tuple spaces*. En fait, il n'y a pas de distinction possible entre l'information véhiculée par les *tuples* et de sa représentation dans un *tuple space* : il n'y a donc pas de moyen de séparer la représentation de l'information de la perception de l'information par les *agents*.

Il n'existe donc pas de façon d'exprimer les politiques de coordination qui ne sont pas directement supportées par les mécanismes standards du *tuple space*, et de les embarquer dans le médium de coordination. Au lieu de cela, la charge de coordination doit généralement (mais improprement) être supportée sur les entitées coordonnées, qui doivent être rendues "*coordination-aware*". Cela signifie que les *agents* ne peuvent pas abstraire les détails de coordination, mais doivent embarquer dans leur code les protocoles d'interaction requis pour implémenter les politiques de coordination nécessaires. Les problèmes d'exécution et de coordination se voient donc involontairement fusionnés dans la conception de l'*agent*, rendant donc cela inutilement complexe.

Garder la représentation et la perception de l'information clairement séparées permet d'outrepasser certaines limites et de s'affranchir d'une conception complexe. Cela permet aux protocoles d'interaction de l'*agent* d'être organisés autour de la perception voulue de cet *agent* de l'espace de communication, indépendamment de son réel contenu en termes de *tuples*. De plus, en mettant proprement en relation la représentation et la perception de l'information, les média de coordination peuvent être en charge de la coordination, en intégrant n'importe quelle loi de coordination, même si elle n'est pas directement supportée par le médium de communication.

Cette caractéristique peut être obtenue en maintenant l'interface standard *tuple space* et, en même temps, en rendant possible l'enrichissement du comportement d'un *tuple space* en termes de transition d'état en réponse à l'apparition d'événements standards de communication. C'est la motivation derrière la notion même de *tuple centre*, qui est donc un *tuple space* enrichi avec la notion de *spécification programmable du comportement*. Un *tuple centre* a exactement la même interface qu'un *tuple space*, mais peut agir de façon complètement différente, vu que son comportement encapsule les règles de coordination gouvernant l'interaction. La spécification du comportement d'un *tuple centre* consiste en une collection de *reaction specifications*, associant n'importe quelle primitive standard de communication des *tuple spaces* à des activités computationnelles spécifiques appelées *reactions*. Les sémantiques des événements de communication ne sont donc plus contraintes à se restreindre au modèle **Linda** (c'est-à-dire, ajouter, lire, et enlever des *tuples*), mais peuvent être rendues aussi complexes que possible suivant les besoins de l'application.

Chaque réaction peut librement accéder et modifier les informations (*tuples*) collectées dans un *tuple centre*, et peut accéder à toute l'information relative au déclenchement des événements de communication (comme l'*agent* exécutant, l'opération requise, le *tuple* impliqué, ...), ce qui la rend donc complètement observable. Chaque événement de communication peut déclencher une multitude de réactions qui sont toutes effectuées sur une seule transition de l'état du *tuple centre*, avant qu'aucun autre événement de communication déclenché par un composant ne soit desservi. En résumé, du point de vue des *agents*, le résultat de l'invocation d'une primitive de communication est la somme des effets de la primitive elle-même et de toutes les réactions qu'elle a déclenchées. Ces effets sont entièrement perçus comme une seule étape de transition de l'état du *tuple centre*. A l'opposé d'un *tuple space* standard, dont les transitions d'état sont contraintes par l'ajout, la lecture ou l'enlèvement d'un seul *tuple*, la transition d'un *tuple centre* peut être rendue aussi complexe que nécessaire. Cela rend possible de découpler la vue que l'*agent* a du *tuple centre* (perçu comme un standard *tuple space*) de l'état réel du *tuple centre*, et de les relier de façon à intégrer correctement les lois de coordination [9].

4.4.2 Spécifications

On définit un *logic tuple centre* comme étant un *tuple centre* où aussi bien les *tuples* de communication que le langage de spécification des réactions sont basés sur la logique (*logic-based*). Le langage **ReSpecT** [9] est un langage *logic-based* dédié à la spécification du comportement de *tuple centres*. Il permet la définition d'exécutions, appelées *reactions*, à l'intérieur d'un *tuple centre* et rend possible l'association des réactions aux événements de communication survenant dans un *tuple centre*. **ReSpecT** a donc aussi bien une partie déclarative que procédurale :

- en tant que langage de spécification, **ReSpecT** permet aux événements de communciation d'être associés de façon déclarative au moyen de *logic tuples* spécifiques, appelés *specification tuples*.
- en tant que langage de réaction, **ReSpecT** permet aux réactions d'être définies de façon procédurale en termes de séquences de *logic reaction goals*, chaque *goal* pouvant réussir ou échouer. Une réaction, dans son ensemble, réussi si tous ses *reaction goals* réussissent, elle échoue dans le cas contraire. Chaque réaction est exécutée séquentiellement avec une sémantique transactionnelle [10] : ainsi, une réaction échouée n'a aucun effet sur l'état du *logic tuple centre*.

9. *Reaction Specification Tuples*.
10. En comparaison aux transactions relatives aux bases de données.

Toutes les *reactions* déclenchées par un événement de communication sont exécutées avant de désservir un autre événement : les *agents* perçoivent donc le résultat du traitement de l'événement de communication et de l'exécution des *reactions* associées dans son ensemble, comme une seule transition de l'état du *tuple centre*. En conséquence, l'effet d'une primitive de communication sur un *logic tuple centre* peut être rendue aussi complexe que nécessaire, suivant les exigences de coordination d'un système *multi-agents*. De manière générale, on peut dire que n'importe quelle loi de coordination peut-être, en principe, encapsulée à l'intérieur d'un *tuple centre* **ReSpecT**.

Un *tuple centre* **ReSpecT** est conceptuellement structuré en deux parties : le *tuple space*, contenant les *tuples* ordinaires de coordination, et le *specification space*, contenant les *tuples* de spécification. Cette distinction suggère deux niveaux d'abstraction au-dessus de l'espace d'interaction de l'*agent* : les points de vue de la *communication* et de la *coordination*. En représentant, à chaque moment, l'état actuel de l'interaction de l'*agent*, l'état de l'espace des *tuples* ordinaires rend disponible le point de vue "*communication*". Au lieu de cela, l'espace des *tuples* de spécification fournit le point du vue "*coordination*", puisque la spécification du comportement d'un *tuple centre* gouverne la communication inter-*agent*, et que les *tuples* de spécification définissent en fait les règles de coordination d'*agent*. D'un autre côté, puisque les deux espaces dans un *tuple centre* **ReSpecT** peuvent être vus comme des collections de clauses logiques unitaires, ils peuvent en quelque sorte être considérés comme des *théories de communication et coordination*. Puisque les *agents* peuvent en principe accéder de manière uniforme aussi bien au *tuple space* qu'au *specification space*, ils peuvent choisir d'adopter, à n'importe quel moment, soit le point du vue "*communication*", soit le point de vue "*coordination*" sur le système *multi-agents* auquel ils appartiennent. Ainsi, l'approche *logic-based* de **ReSpecT** permet aux *agents* de raisonner sur le statut et le comportement du système en prenant en considération aussi bien la théorie de communication que la théorie de coordination, et d'habiliter les *agents* avec la capacité de changer les lois de coordination en agissant sur l'espace de spécification [8].

Le comportement d'un *tuple centre* peut être défini ou modifié dynamiquement (pendant l'exécution) via les *agents* au moyen de l'opération `set_spec` (définie dans le tableau 4.2). Ce type de modèle est donc adéquat pour les systèmes ouverts devant changer, adapter ou accorder leurs politiques de coordination lors de l'éxécution du système. L'inspection du comportement d'un *tuple centre* est réalisée via la commande `get_spec`, qui peut aussi être utilisée dynamiquement.

Un programme **ReSpecT** prend la forme d'un ensemble de réactions :

reaction(Event, (Body)).

où

Event est un événement de communication étendu, incluant l'exécution des primitives basiques de coordination (`out`, `in`, `inp`, `rd`, `rdp`), et l'exécution (réussie) de quelques primitives **ReSpecT** (`out_r`, `in_r`, `rd_r`, `no_r`) qui seront explicitées plus loin dans le document.

Body est une séquence de prédicats **ReSpecT**. Les prédicats principaux sont listés dans le tableau 4.4 et permettent d'inspecter et de changer le contenu d'un ensemble de *tuple* en insérant, prélevant et lisant les *tuples* respectifs. Aussi, quelques prédicats de **Prolog** et quelques fonctions sont disponibles pour rendre plus facile la manipulation et la comparaison des symboles et des nombres.

Il est à noter que les réactions sur les évènements `in` et `rd` sont déclenchées deux fois : la première (appelé *pre stage*) lorsque la primitive de coordination est émanée de l'*agent*, et la deuxième (appelé *post stage*) lorsqu'un résultat (un *tuple*, donc) est éventuellement renvoyé à l'*agent* faisant la requête. La syntaxe complète du langage **ReSpecT** est présentée dans le listing 4.3.

Spec : := *Reaction**
Reaction : := reaction(*Event*, (*Body*)).
Event : := *CommunicationEvent* | *InternalEvent*
CommunicationEvent : := out(T) | in(TT) | inp(TT) | rd(TT) | rdp(TT)
InternalEvent : := out_r(T) | in_r(TT) | rd_r(TT)
Body : := *Predicate*{ , *Body*}
Predicate : := *BasicPredicate* | *ExtendedPredicate*
BasicPredicate : :=
out_r(T) | in_r(TT) | rd_r(TT) | no_r(TT) |
pre | post |
current_tuple(TT) | current_agent(TT) | current_op(TT) |
current_tc(TT)
ExtendedPredicate : :=
X is *Expression* |
BooleanExpression |
out_tc(*TC*, *T*) |
spawn(*AgentId*, *AgentType*, [{*ArgList*}]) |
current_time(TT)
Expression : := *AritmeticExpression*
BooleanExpression : := *AritmeticComparison*
AgentType : := java(*AgentClassName*) | prolog(*TheoryFileName*)
ArgList : := *T*{ , *ArgList*}

TABLE 4.3 – Syntaxe de ReSpecT

4.4.2.1 Idiomes de base

Les constructions basiques qui sont typiquement adoptées dans le développement d'algorithmes peuvent être réalisées dans le langage **ReSpecT** en arrangeant de manière adéquate les réactions. On considère dans les exemples qui suivent, les idées et les idiomes de base pour réaliser des sélections, des itérations et de la récursivité.

Sélections

La construction d'une sélection peut être réalisée par un ensemble de réactions qui ont le même `Event` déclencheur, mais contenant dans la première partie de leur `Body`, des prédicats satisfaits sur des conditions exclusives, de manière à n'avoir qu'une seule réaction possible à la fois.

Dans l'exemple ci-dessous, on suppose que l'on veuille d'abord effectuer une opération ***A*** en réaction à l'événement ***E***, et puis exécuter l'opération ***B*** ou ***C*** suivant la présence d'un certain *tuple* ***T*** dans le *tuple centre*.

```
reaction( E, (
    A,
    out_r(check(T))) ).

reaction( out_r(check(T)), (
    rd_r(T),
    B) ).

reaction( out_r(check(T)), (
    no_r(T),
    C) ).
```

Dans l'exemple qui suit, on suppose que l'on veuille insérer un *tuple* `is_greater(X,Y)` lorsque le *tuple* `compare(X,Y)` est inséré dans le *tuple centre* avec X plus grand que Y. Dans le cas

contraire, on insère le *tuple* `is_less(X,Y)`. On prend pour hypothèse que X et Y sont définis comme étant des nombres.

```
reaction(out(compare(X,Y)), (
    X > Y,
    out_r(is_greater(X,Y)) )).

reaction(out(compare(X,Y)), (
    X < Y,
    out_r(is_less(X,Y)) )).
```

Récursivité et itérations

La récursivité en **ReSpecT** consiste en des réactions se délenchant elle-mêmes. Ce qui est aussi le seul moyen de réaliser des itérations, tout en exploitant les sélections de manière adéquate. L'exemple du calcul de la factorielle d'un nombre (entier) permet facilement de mettre en évidence la récursivité et l'itération [11].

```
reaction(out(factorial(N,_)), (
    in_r(factorial(N,_)),
    out_r(fact_loop(1,N,1)) )).

reaction(out(fact_loop(N,N,F)), (
    in_r(fact_loop(N,N,F)),
    out_r(factorial(N,F)) )).

reaction(out(fact_loop(I,N,F)), (
    in_r(fact_loop(I,N,F)),
    N > I,
    I1 is I + 1,
    F1 is F * I1,
    out_r(fact_loop(I1,N,F1)) )).
```

La stratégie consiste à transformer le *tuple* `fact_loop(I,N,F)` jusqu'au moment où les deux premiers arguments coïncident, le premier argument étant l'index des nombres entiers allant de ***1*** à ***N*** (inclus). Lorsque cet index ***I*** est égal au nombre ***N***, alors le nombre ***F*** - qui se transformait au fur et à mesure que l'index s'incrémentait - contient la factorielle du nombre ***N***.

La récursivité se trouve dans la troisième réaction : elle est déclenchée par un événement (l'insertion de `fact_loop`) qui est généré par la réaction elle-même.

FIGURE 4.10 – Logo ReSpecT

11. On remarquera dans l'exemple l'utilisation de la variable anonyme notée '_'.

Accès et modification du tuple space	
out_r(T)	Réussi toujours et, comme effet, le *logic tuple* **T** est inséré dans le *tuple set*.
in_r(TT)	Réussi si un *logic tuple* correspondant au template **TT** est présent dans le *tuple set* et, comme effet, le *tuple* est enlevé et unifié avec **TT**. Sinon, le prédicat échoue.
rd_r(TT)	Réussi si un *logic tuple* correspondant au template **TT** est présent dans le *tuple set* et, comme effet, le *tuple* est unifié avec **TT**. Sinon, le prédicat échoue.
no_r(TT)	Réussi si aucun *logic tuple* correspondant au template **TT** n'est présent dans le *tuple set*. Sinon, le prédicat échoue.
Information sur l'événement de communication	
pre	Réussi si l'évènement déclenchant la réaction est une requête (**in** ou **rd**) dans le *pre stage*, c'est-à-dire, au moment de la requête par l'*agent*.
post	Réussi si l'évènement déclenchant la réaction est une requête (**in** ou **rd**) dans le *post stage*, c'est-à-dire, au moment du renvoi du résultat à l'*agent* demandeur.
success	Réussi si l'évènement déclenchant la réaction est un **inp** ou un **rdp** réussi.
failure	Réussi si l'évènement déclenchant la réaction est un **inp** ou un **rdp** échoué.
current_agent(A)	Réussi si **A** s'unifie avec l'identifiant de l'*agent* qui a déclenché l'événement de communication courrant.
current_tuple(T)	Réussi si **T** s'unifie avec le *tuple* impliqué par l'événement de communication courrant.
current_op(Op)	Réussi si **Op** s'unifie avec le descripteur de l'opération qui a produit l'événement de communication courrant.
current_tc(N)	Réussi si **N** s'unifie avec l'identifiant du *tuple centre* exécutant la computation.
current_time(Ti)	Réussi si **Ti** s'unifie avec le temps de la machine virtuelle du *tuple centre*.
out_tc(TC, T)	Réussi si **TC** est un nom de *tuple centre* valide et atteignable (qui peut résider dans un noeud différent). Ce prédicat a le même effet que (et est aussi modélisé comme) l'opération **out** sur le *tuple centre* spécifié : le *tuple* **T** est inséré dans le *tuple centre* **TC**.
spawn(AgentID, AgentType, ArgList)	Réussi si **AgentID** est un identifiant d'*agent* valide, **AgentType** indique un type d'*agent* valide et **ArgList** est une liste de *tuples*. Le prédicat a comme effet d'engendrer un nouvel *agent* avec les caractéristiques spécifiées, en passant les arguments nécessaires.

TABLE 4.4 – Principaux prédicats du langage ReSpecT

4.5 Approche organisationnelle

4.5.1 Agent Coordination Context

Des études récentes sur l'histoire des sociétés humaines suggèrent que le rôle de l'*environnement* doit être pris explicitement en considération dans le but de comprendre l'évolution des individus et des groupes dans n'importe quel cadre non-trivial. La notion de *contexte* est bien connue et pertinente pour plusieurs domaines de recherche tels que le langage naturel, la philosophie, la logique et l'intelligence artificielle. Dans ces domaines, les contextes sont typiquement utilisés pour modéliser l'effet de l'environnement (dans son sens le plus général, incluant les interprétations spatiale et temporelle du terme) sur la communication survenant parmi les entités actives, telles que les humains ou les agents artificiels.

En généralisant sur la notion - récemment introduite - de coordination dépendante du contexte (*context-dependent coordination*), on suggère que la notion de *agent coordination context* est un moyen de modéliser et de façonner l'environnement dans les systèmes d'*agents*. Cette notion est appelée à jouer, dans un certain sens, le même rôle à l'intérieur des systèmes d'*agents* que la notion d'interface d'objet dans les systèmes d'objets : à savoir, fournir une discipline pour l'interaction et certains modèles pour gérer le contrôle. Puisque, grossièrement parlant, le contrôle se situe en dehors des objets mais à l'intérieur des *agents*, un *agent coordination context* est en fait destiné à fournir une sorte de "description des limites" comme le fait l'interface d'un objet - mais de l'intérieur vers l'extérieur (d'un *agent*), plutôt que de l'extérieur vers l'intérieur (d'un objet). Cependant, en tant qu'interface, un contexte de coordination fournit du découplage entre les *agents* et leur environnement, de telle manière que l'*agent* puisse, en principe, être conçu et développé indépendamment des autres *agents*, ressources et services peuplant l'environnement du système *multi-agents*.

Plus précisément, un contexte de coordination devrait :
- travailler comme un modèle pour l'environnement de l'*agent*, en décrivant l'environnement où l'*agent* peut interagir, et
- permettre et gouverner les interactions entre l'*agent* et l'environnement, en définissant l'espace des interactions admissibles d'*agent*

En tant qu'outil pour modéliser l'environnement, l'*agent coordination context* peut servir un objectif double, selon le point du vue que l'on prend. D'un point de vue de l'*agent* (plus généralement appelé le point du vue *subjectif*), un *agent coordination context* devrait fournir aux *agents* une représentation adéquate de l'environnement où ils résident, interagissent et communiquent. Une telle représentation devrait manifestement inclure quelques notions de *localité* (spatiale et temporelle) - puisqu'un environnement global et synchrone n'est pas réaliste dans les scénarios d'applications actuelles -, définissant implicitement le "*où*" et le "*comment*" des interactions d'*agents*. Il est évident qu'un contexte de coordination devrait contenir toutes les informations relatives aux entités (*agents*, services, ressources) avec lesquelles un *agent* peut interagir, en même temps que la description de la voie admissible d'interaction. Cela part du postulat qu'un langage de représentation de contexte est défini et utilisé : cela peut être un langage logique du premier ordre, basé sur le XML, ou qu'importe - tant qu'il est assez expressif pour décrire complètement l'environnement de l'*agent* et toutes les interactions admissibles des *agents*.

Du point de vue du concepteur humain (plus généralement appelé le point de vue *objectif*), un *agent coordination context* devrait fournir un *framework* pour exprimer l'interaction à l'intérieur d'un système *multi-agents* (dans son ensemble). Plus précisément, les contextes de coordination définissent l'espace d'interaction d'un système *multi-agents*, à savoir, les interactions admissibles survenant parmi les *agents* d'un système *multi-agents*, et entre les *agents* d'un système *multi-agents* et l'environnement d'un système *multi-agents*. En particulier, dans la mesure où la communication inter-*agent* est concernée, les contextes de coordination modélisent comment l'environnement affecte l'interprétation des actes de communication des *agents*.

En tant qu'outil pour permettre et gouverner l'interaction des *agents* avec l'environnement, un *agent coordination context* peut aussi servir un objectif double. Du point de vue de l'*agent*,

le contexte de coordination permet en principe aux *agents* de percevoir l'espace où ils agissent et interagissent, raisonner sur l'effet de leurs actions et leurs communications, et éventuellement affecter l'environnement pour accomplir leurs propres buts. Du point de vue d'un ingénieur (humain, donc), les contextes de coordination permettraient aux ingénieurs d'encapsuler les règles pour gouverner les applications construites commes des systèmes d'*agents*, d'arbitrer les interactions parmi les *agents* et l'environnement, et éventuellement de les affecter de manière à changer le comportement global de l'application de façon incrémentale et dynamique. Les contextes de coordination permettent alors à une forme de *coordination normative* d'être respectée, contraignant l'espace d'interaction de l'*agent* de l'étape de conception au temps d'exécution (*run-time*). Il est à remarquer qu'une infrastructure "hôte" ayant la capacité de faire en sorte que les comportements préjudiciables pourraient être évités, et que seules les actions admissibles pourraient être entreprises par les *agents*, serait à même de fournir des espaces ouverts (*open spaces*) où les *agents* peuvent agir de leur propre chef et suivant leur volonté, libres d'accomplir leurs propres buts sans entraver, à priori, leur modèle et leur comportement. De ce point de vue, alors, la *coordination normative* semble être une sorte de pré-condition, plutôt qu'un obstacle, à l'autonomie de l'*agent*.

Cette réflexion implique aussi un support approprié lors de l'exécution (*run-time*), incarnant les contextes de coordination commes des abstractions dynamiques fournies par l'infrastructure de l'*agent*. Aussi, par leur nature même, les *agent coordination contexts* peuvent être utilisés pour servir de médiateur entre les besoins d'une application d'un système *multi-agents* et son hôte, en modélisant les infrastructures des *agents* (leur structure et leur organisation, les ressources et les services qu'elles rendent disponibles aus *agents*) en termes de contextes de coordination. Les contextes de coordination, lorsque convenablement supportés en tant qu'abstractions dynamiques, devraient être dynamiquement *configurables* et *inspectables* par les *agents* aussi bien que par les humains.

- La ***configurabilité*** permettrait à un système *multi-agents* d'évoluer lors de l'exécution, en adaptant de façon adéquate son comportement aux changements. Ce qui est une des exigences majeures dans les scénarios complexes et fortement dynamiques.
- L'***inspectabilité*** permettrait aussi bien aux humains qu'aux *agents* intelligents de raisonner sur les lois existantes de coordination telles que représentées et incarnées à l'intérieur des contextes de coordination, et éventuellement de les changer en reconfigurant proprement les contextes de coordination suivant les nouveaux besoins de l'application.

Enfin, puisque les modèles de coordination ont été reconnus comme une base pour imposer/appliquer les règles sociales, et que ces contextes sont facilement interprétés comme des concepts sociaux, les contextes de coordination ont le potentiel d'être exploités pour l'ingénierie de l'ordre social au sein des sociétés d'*agents* [7].

4.5.1.1 La métaphore de la salle de contrôle

Il est évident que la notion d'*agent coordination context* contient beaucoup d'éléments et de détails qui doivent être définis et rendus plus précis avant de devenir réellement implémentable et utilisable. Les cas des *agent coordination contexts* peuvent être donnés soit en termes d'exemple conceptuel, soit en termes de modèle de coordination où les aspects d'un contexte de coordination peuvent être dûment conçus à un niveau approprié d'abstraction.

La métaphore de la salle de contrôle, fournit un exemple conceptuel d'un *agent coordination context*. Selon cette métaphore, un *agent* pénétrant dans un nouvel environnement est affecté à sa propre salle de contrôle, qui est le seul moyen qu'il a de percevoir l'environnement, aussi bien que le seul moyen qu'il a d'interagir. Dans cette salle, les entrées (*inputs*) admissibles d'*agent* sont représentées par des lumières (entrées discrètes) et des écrans (entrées continues), alors que les sorties (*outputs*) admissibles d'*agent* sont représentées par des boutons (sorties discrètes) et des caméras (sorties continues). Dès lors, par exemple, une communication continue bidirectionnelle avec un autre *agent* peut être exploitée au travers d'une paire écran-caméra.

Combien de "dispositifs" d'entrées et de sorties sont disponibles pour un *agent*, de quelle sorte, pour combien de temps - c'est ce que définit la ***configuration*** de la salle de contrôle, qui est, en fait, le contexte spécifique de coordination de l'agent (*specific agent coordination context*).

Une telle configuration, représentant une sorte d'interface de l'environnement pour l'*agent*, est à n'importe quel moment prescriptive (ou normative), en ce sens qu'elle décrit complètement toutes les interactions admissibles pour un *agent*. En fait, la configuration initiale de la salle de contrôle est sujette à des négociations préliminaires entre l'*agent* et le noeud hôte. Ensuite, elle peut être dynamiquement modifiée selon les changements dans les besoins de l'*agent* ou du noeud hôte fournissant le contexte de coordination. Par exemple, dû à un manque soudain de ressources, une salle de contrôle peut être réduite par son infrastructure hôte à un contexte vide (en d'autres mots, aucune interaction permise), forçant implicitement l'*agent* à "déménager" afin d'être capable d'effectuer certaines actions éloquentes. On peut aussi penser à associer un coût (par unité de temps, par utilisation) à chaque dispositif rendu disponible par une configuration, de sorte que les négociations pour la configuration initiale de la salle de contrôle prendrait aussi en compte l'enjeu économique de la communication.

4.5.2 Le modèle de coordination de TuCSoN

Un modèle de coordination tel que défini précédemment fournit un *framework* et une technologie pour façonner et gouverner l'espace d'interaction d'un *agent*. Un exemple éloquent de modèle et de technologie de coordination est **TuCSoN**.

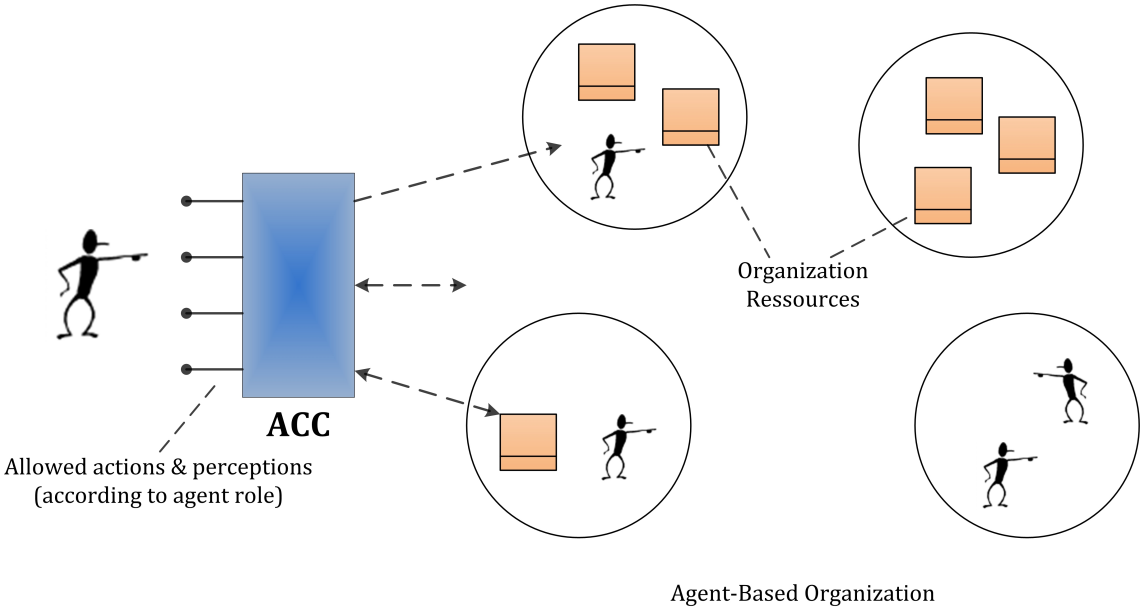

FIGURE 4.11 – Approche organisationnelle de TuCSoN

En termes de contexte de coordination, **TuCSoN** fournit à chaque *agent* une vue de l'espace d'interaction qui comprend une collection de média de coordination, c'est-à-dire les *tuple centres* - mais ce n'est pas nécessairement limité à cela. D'une part, en réalité, l'infrastructure **TuCSoN** n'est pas normative du tout : n'importe quel schéma d'interaction autre que les *tuple centres* est en principe disponible aux *agents*. Par exemple, deux *agents* peuvent choisir d'interagir directement et de contourner le *tuple centre*. D'autre part, puisque les *tuple centres* sont fournis en tant que *services de coordination* par l'infrastructure **TuCSoN**, le choix délibéré des *agents* de les exploiter est suffisant pour limiter et contraindre leur interaction : tant que les *agents* interagissent au travers des *tuple centres*, ils suivront les lois de coordination que les *tuple centres* intègrent. Aussi, le contexte de coordination fournit par **TuCSoN** est en général inspectable et configurable dynamiquement par les humains et les *agents* (intelligents) : les humains peuvent accéder et modifier les *tuple centres* au travers des *inspecteurs*, rendus disponibles par l'infrastructure en tant qu'outils de développement et de déploiement, alors que les *agents* sont munis d'un ensemble de primitives spéciales pour inspecter et changer les spécifications du comportement du *tuple centre*. Puisque les *tuple centres* encapsulent aussi bien l'état de communication que les lois de coordination en tant que clauses

logiques du premier ordre, ils peuvent, en principe, être exploités par les humains et les *agents* pour surveiller un système *multi-agents* au cours du temps, ainsi que pour éventuellement changer dynamiquement le comportement du système *multi-agents* en affectant de manière adéquate leur spécification **ReSpecT** de comportement.

4.5.2.1 Extension du modèle TuCSoN

Le concept abstrait d'*agent coordination context*[12] a été mis en pratique grâce à son intégration dans l'infrastructure de coordination de **TuCSoN** pour les systèmes *multi-agents*. **TuCSoN** a été conçu au départ comme une infrastructure purement "*de coordination*" pour les systèmes basés sur des *agents* (mobiles), de manière orthogonale au modèle de computation d'*agent*. L'infrastructure a été étendue pour couvrir les aspects de sécurité en synergie avec la coordination et, maintenant, le concept d'ACC est utilisé pour intégrer un modèle d'organisation basé sur la notion de rôle (*role-based organization model*) et pour étendre la sécurité au travers d'une approche RBAC (*Role-Based Access Control*[13]). Le modèle de l'infrastructure peut être résumé en décrivant les quatre principaux aspects connexes : le modèle de coordination, le modèle de topologie, le modèle d'organisation et le modèle de sécurité [12].

- Le modèle de coordination de **TuCSoN** est basé sur l'approche "coordination en tant que service", qui promeut l'exploitation de services/artéfacts de coordination d'usage général (les *tuple centres*) qui peuvent être dynamiquement personnalisés (programés) afin de supporter la coordination des *agents*.
- Du point de vue de la topologie, les artéfacts de coordination de **TuCSoN** sont collectés dans les noeuds de l'infrastructure, distribués sur le réseau, organisés en domaines "*articulés*". Un domaine est caratérisé par un noeud "passerelle" (*gateway node*) et un ensemble de noeuds appelés "lieux" (*places*). Sans rentrer dans les détails, un "lieu" est destiné à héberger les *tuple centres* d'applications spécifiques, tandis qu'un noeud "passerelle" est déstiné à héberger les *tuple centres* utilisés pour l'administration de domaine, maintenant l'information sur les "lieux". Un "lieu" peut appartenir à différents domaines, et peut lui-même être une "passerelle" pour un sous-domaine.
- Du point de vue de l'organisation, **TuCSoN** adopte un modèle organisationnel basé sur la notion de rôle. Chaque système (ou application) séjournant dans l'infrastructure est structuré comme une organisation composée d'un ensemble de sociétés (*societies*), où des rôles spécifiques sont définis. Un *agent* participe à l'organisation (au système) en jouant toujours un ou plusieurs rôles, dans la même ou dans différentes sociétés. La participation est entièrement dynamique : les *agents* peuvent dynamiquement entrer et sortir d'une organisation, activant différents ensembles de rôles au travers d'une re-négociation de l'ACC. Chaque organisation est mappée sur un domaine, et cela comprend donc au moins une "passerelle" et un ensemble (non vide) de "lieux". Les informations dynamiques concernant les structures et les règles de l'organisation (sociétés disponibles, rôles, lois gouvernant l'organisation) sont stockées en tant que *logic tuples* dans un *tuple centre* spécifique noté `$ORG(OrgID)`, où `OrgID` est l'identifiant de l'organisation.
- Le modèle de sécurité est étroitement lié aux modèles de topologie et d'organisation. La notion de rôle rend possible d'adopter un contrôle d'accès basé sur le rôle, de manière analogue aux modèles RBAC, que l'on trouve dans le management de la sécurité de systèmes complexes, étendant les structures basiques d'authentification et d'autorisation. Les rôles déterminent l'ensemble des actions et de protocoles d'interaction que l'*agent*, concerné par ce rôle, est autorisé à exécuter.

12. L'acronyme ACC sera utilisé par la suite pour désigner l'*Agent Coordination Context*.

13. Les modèles RBAC, considérés à l'heure actuelle comme étant l'approche la plus efficace pour la modélisation de la sécurité dans les systèmes d'information et dans les organisations complexes, peuvent être dûment modélisés avec la notion d'*agent coordination context*.

4.5.2.2 Sémantique du contexte de coordination d'agent de TuCSoN

Un *agent coordination context* est défini par une *Interface*, un *Contract* et un *Contract State*. L'*Interface* définit l'ensemble des opérations que l'*agent* peut utiliser pour agir à l'intérieur d'une organisation **TuCSoN**. Le *Contract* est une description des relations entre l'*agent* et l'organisation, et en particulier, de la politique adoptée par l'ACC gouvernant les actions et les protocoles d'interaction de l'*agent*. Le *Contract State* est une description de l'état d'exécution de l'ACC, concernant l'évolution des protocoles d'interaction en cours. Les informations du *Contract* et du *Contract State* sont destinées à être inspectables afin de mettre en avant le raisonnement de l'*agent* à propos de sa position actuelle à l'intérieur de l'organisation.

ACC Interface. La structure du modèle abstrait de l'*Interface* est présenté dans le listing 4.5. La notation **Prolog** a été adoptée pour décrire les commandes et les fonctionnalités de l'interface, sous la forme de prédicats logiques. Les signes `+`, `-` et `?` signifient respectivement un paramètre d'entrée, de sortie et d'entrée/sortie du prédicat. Toutefois, cette règle pour le signe `?` ne s'applique pas pour la description de l'*ACCAction*, où il est utilisé en tant qu'opérateur binaire (comme présenté dans la section 4.3.2 en page 19).

ACCInterface : :=
doAction(+*ACCAction*, -*ActionID*) |
isActionCompleted(+*ActionID*, ?{true|false}) |
isActionFailed(+*ActionID*, ?{true|false}) |
action(+*ActionID*, ?*ACCAction*) |
actionResult(+*ActionID*, ?*ACCActionCompleted*) |
getACCSpec(-*ACCSpec*) |
getACCState(-*ACCState*)
ACCAction : :=
CoordinationOp |
Tid ? *CoordinationOp* |
Tid@Node ? *CoordinationOp*
CoordinationOp : :=
out(T) | in(TT) | rd(TT) | inp(TT) | rdp(TT) |
set_spec(T) | get_spec(TT)

TABLE 4.5 – Syntaxe de l'ACC Interface

La requête `doAction` est utilisée pour exécuter une action ACC, qui est l'exécution d'une opération sur un *tuple centre* spécifique. Si l'action n'est pas permise par la politique de l'ACC, une exception est générée, empêchant l'exécution de l'action ; sinon, l'action est exécutée et une référence d'action est renvoyée à l'*agent*. Les opérations de coordination sont les primitives de coordination des *tuple centres* pour insérer, lire et retirer des *tuples* et pour inspecter/changer la spécification du comportement d'un *tuple centre*. `isActionCompleted` vérifie si une action exécutée s'est terminée, ou si sa complétion est toujours en attente : c'est le cas, par exemple, de l'invocation d'une primitive `in` ou `rd` sur un *tuple centre* sans *tuple* correspondant. `isActionFailed` vérifie plutôt si des échecs possibles sont survenus après l'exécution de l'action et avant sa complétion. La requête `action` est fournie pour obtenir l'action ACC correspondante à l'identifiant donné. La requête `actionResult` est prévue afin d'inspecter les informations potentielles issues de la complétion d'une action. Elle est utilisée dans le cas où les actions ont atteint leur complétion. Il est important de noter qu'il n'y a pas d'action bloquante : `doAction` retourne immédiatement même lorsqu'elle invoque les opérations comme `in` et `rd` - qui sont des opérations bloquantes dans le modèle basique **Linda**. Enfin, l'interface fournit des moyens d'inspecter l'*ACC Contract* et l'*ACC Contract State* (grâce à `getACCSpec` et `getACCState` respectivement).

ACC Contract. Le contrat contient la description des relations établies entre l'*agent* et l'organisation, encodé en langage **Prolog** sous la forme d'une théorie logique. L'information inclut le nom de l'organisation délivrant l'ACC (`organisation`(*ID*)), les rôles activement joués par l'*agent* (`role`(*ID*, *SocietyID*)), la date d'obtention de l'ACC (`release_date`(*Date*)), et, potentiellement, une période de validité (`validity`(*DateFrom*, *DateTo*) ou en mentionnant une période de temps `validity`(*LeaseTime*)).

Plus important, le contrat contient la politique que l'ACC utilise pour établir si l'action d'un *agent* peut être exécutée ou non. Cette politique d'ACC est obtenue en composant les politiques des rôles individuels, qui sont stockées en tant que *logic tuples* à l'intérieur du *tuple centre* `$ORG` de la passerelle admettant l'*agent*.

La théorie décrivant une politique de rôle est composée par un ensemble de règles `can_do` :

`can_do`(*CurrentState*, *Action*, *NextState*) :- *Conditions*.

Cette règle signifie que l'action `Action` peut être exécutée dans l'état de rôle `CurrentState` si les conditions `Conditions` sont vérifiées et, dans ce cas, que le prochain état de rôle est `NextState`. Le concept d'état de rôle est utilisé comme moyen d'exprimer facilement les protocoles d'interactions ; n'importe quel terme **Prolog** - même partiellement spécifié - peut être utilisé pour indiquer l'état de rôle. Par défaut, l'état de départ est indiqué par l'atome `init`. `CurrentState` et `NextState` peuvent être mis de côté en utilisant le symbôle **Prolog** " _ " : omettre l'information `CurrentState` affirme la validité de la règle pour chaque état possible ; omettre l'information `NextState` garde l'état courant comme prochain état. Le paramètre `Action` signifie l'action de l'*agent* telle que définie dans l'*ACC Interface*.

En réalité, le terme `Conditions` peut aussi contenir des prédicats pré-construits utiles pour décrire des politiques sensibles au contexte (en ce qui concerne l'heure locale, l'endroit actuel ainsi que l'identité et les positions des *agents*). Parmis les prédicats, on mentionnera : `agent_id`(-*ID*) (qui renvoie l'identité de l'*agent* détenant l'ACC), `local_node`(-*Node*) (qui renvoie le noeud hébergeant l'*agent*), `session_time`(-*TimeInMillis*) (qui renvoie le nombre de millisecondes écoulées depuis l'obtention de l'ACC).

Donc, en donnant la politique d'ACC, une action est admise si et seulement si il y a une règle `can_do` qui est valable pour cette action ; en d'autres mots, une action `Action` est permise si le but `can_do`(*CurrentState*, *Action*, *NextState*) peut être démontré, compte tenu de la théorie. Par exemple, une règle de type `can_do`(_, _, _). signifie que chaque action est permise. Une autre règle de type `can_do`(_, *Action*, _). indique que l'action `Action` est toujours exécutée. En spécifiant le corps (*Body*) des règles, il est possible d'exprimer aussi des actions interdites spécifiques : `can_do`(_, Act, _) :- `not`(Act=*Action*). signifie que chaque action peut être éxécutée sauf celles correspondant au modèle `Action`.

Comme simple exemple de politique ACC, considérons un service de messagerie dans lequel le *tuple centre* `msg_box` est utilisé de telle manière à échanger des messages représentés par des *tuples* `msg`(*DestID*,*Content*). Une politique convenable d'ACC pour les utilisateurs serait alors :

`can_do`(_, `msg_box` ? `out`(`msg`(*DestID*, *Content*)), _).
`can_do`(_, `msg_box` ? `in`(`msg`(*DestID*, *Content*)), _) :- `agent_id`(*DestID*).

Cette politique permet à un utilisateur d'envoyer des messages à n'importe quel autre utilsateur, mais de recevoir seulement les messages qui lui sont destinés.

A cet exemple, on peut ajouter la notion de rôle afin d'obtenir une théorie complète pour la politique de l'ACC.

```
organisation(fundp).
role(employee,mailbox_service).
can_do(_, msg_box ? out(msg(DestID, Content)), _).
can_do(_, msg_box ? in(msg(DestID, Content)), _) :- agent_id(DestID).
```

Ici, on définit un rôle `employee` pour un *agent* au sein de l'organisation `fundp` composée de plusieurs sociétés (telles que définies dans la section 4.5.2.1 en page 32), dont l'une est identifiée par `mailbox_service` et s'occupe du système de messagerie.

Un exemple d'ACC avec une action temporisée est :

```
can_do(_, Action, _) :- session_time(ST), ST < T.
```

où l'action `Action` peut seulement être exécutée pendant une période de temps `T`.

Enfin, un exemple montrant une politique sensible au contexte est :

```
can_do(_, Tid ? out(_), _) :- local_node(Node).
```

Cette règle fait valoir que seulement les *agents* situés dans le noeud `Node` peuvent insérer des *tuples* dans le *tuple centre* `Tid`.

ACC Contract State. Décrit l'état dynamique de l'ACC, en termes d'informations contextuelles liées à la localisation de l'*agent* (aussi bien spatiale que temporelle), et l'état logique de l'interaction en cours. Aussi, ces informations sont encodées en tant que théorie **Prolog**, composée d'un ensemble de faits.

L'ontologie actuelle inclut `current_time`(*Time*) et `current_node`(*Node*), qui décrivent respectivement l'heure courante et le noeud courant de l'ACC (et donc, de l'*agent* l'utilisant), et `current_state`(*RoleState*), qui décrit l'état logique courant tel que défini pour la politique d'ACC.

4.5.3 Agent Java

TuCSoN est donc fourni avec une API qui permet d'exploiter les services des applications écrites en **Java** (ou en **tuProlog**[14]). Afin d'accéder à l'infrastructure TuCSoN, un ACC (*Agent Coordination Context*), tel que défini précédemment, doit d'abord être obtenu. La partie qui suit va permettre de comprendre comment un *agent* **Java** peut obtenir un ACC et invoquer des primitives sur les *tuple centres* appartenant à l'organisation.

Les *packages* fondamentaux requis par n'importe quel programme **Java** souhaitant exploiter l'infrastructure **TuCSoN** sont :

- `alice.tucson.api` : Ce *package* inclut les classes fournissant l'accès à l'infrastructure **TuCSoN**, les classes représentant l'identifiant du *tuple centre* (`TupleCentreId`) et l'identifiant de l'agent (`AgentId`).
- `alice.logictuple` : Ce *package* inclut les classes représentant le langage de communication de **TuCSoN**, c'est-à-dire les *logic tuples*.

L'*agent coordination context* est représenté par l'interface `TucsonContext` (listing 4.1). Il fournit les primitives basiques de coordination telles que décrites dans le tableau 4.2. Dans la version de **TuCSoN** utilisée (1.4.5), un ACC peut seulement être accédé par un *thread* de contrôle : les *agents* composés de multiples *threads* doivent donc obtenir de multiples ACC.

14. **tuProlog** est l'environnement **Prolog** de **TuCSoN**. Il ne sera pas détaillé dans ce document.

```
package alice.tucson.api;

import alice.logictuple.*;

public interface TucsonContext {

  void out(TupleCentreId tid, LogicTuple t)
       throws  OperationNotAllowedException,
               UnreachableNodeException;

  LogicTuple in(TupleCentreId tid, LogicTuple t)
       throws  OperationNotAllowedException,
               UnreachableNodeException;

  LogicTuple rd(TupleCentreId tid, LogicTuple t)
       throws  OperationNotAllowedException,
               UnreachableNodeException;

  LogicTuple inp(TupleCentreId tid, LogicTuple t)
       throws  OperationNotAllowedException,
               UnreachableNodeException;

  LogicTuple rdp(TupleCentreId tid, LogicTuple t)
       throws  OperationNotAllowedException,
               UnreachableNodeException;

  void setSpec(TupleCentreId tid, String spec)
       throws  OperationNotAllowedException,
               UnreachableNodeException,
               InvalidSpecificationException;

  String getSpec(TupleCentreId tid)
       throws  OperationNotAllowedException,
               UnreachableNodeException;

  void exit()
       throws  OperationNotAllowedException;

}
```

Listing 4.1 – TucsonContext Interface

Quelques points méritent d'être soulevés :

- L'exception `OperationNotAllowedException` est levée si l'action effectuée n'est pas permise par l'ACC.
- La méthode `exit` est utilisée pour libérer l'ACC obtenu.
- Le *logic tuple* retourné par les primitives de coordination `in`, `inp`, `rd`, `rdp` se réfère au *logic tuple* qui a été fourni en argument, après avoir été sujet à l'unification impliquée dans la primitive. Ces primitives de coordination agissent directement sur le *logic tuple* fourni en tant qu'argument (*tuple template*), exploitant le service d'unification qu'elles fournissent.

La description de l'ACC est représentée par la classe `ContextDescription` présentée (partiellement) au listing 4.2.

```
package alice.tucson.api;

import alice.tucson.api.*;

public class ContextDescription implements java.io.serializable {

  public ContextDescription(String theory);

  public ContextDescription(java.util.Properties p);

  ...
```

```
}
```

Listing 4.2 – ContextDescription Class

Le contexte de coordination est représenté par la classe `Tucson` (présentée partiellement au listing 4.3), fournissant les services (statiques) basiques pour négocier/acquérir un *agent coordination context*. Cette classe agit comme un moyen de découvrir l'infrastructure.

```
package alice.tucson.api;

import alice.logictuple.*;

public class Tucson {

  public static TucsonContext getContext(AgentId id)
      throws ContextNotAvailableException;

  public static boolean isActive(InetAddress node);

  public static String getVersion();

  ...
}
```

Listing 4.3 – Tucson Class

4.5.3.1 Un exemple simple : "Hello World"

Le petit programme qui suit (listing 4.4) va permettre de bien comprendre comment créer et utiliser un *agent* qui va interagir avec l'environnement **TuCSoN**. Tel que décrit dans les sections précédentes, le programme va créer un *agent*. Ce dernier va obtenir un *agent coordination context* qui lui permettra d'invoquer des primitives de coordination.

```
package tucsonTestImpl;

import alice.tucson.api.*;
import alice.logictuple.*;

public class TuCSoNTest {

    public static void main(String[] args) throws Exception {

        /* Step 1 - Create Agent ID */
        AgentId aid = new AgentId("user");

        /* Step 2 - Obtain Agent's default ACC */
        TucsonContextSynch ctx = new TucsonContextSynch(Tucson.getContext(aid));

        /* Step 3 - Create Tuple Centre ID */
        TupleCentreId tcid = new TupleCentreId("test");

        /* Step 4 - Create a Tuple */
        LogicTuple t = new LogicTuple("p", new Value("hello world"));

        /* Step 5 - Send the Tuple into the Tuple Centre */
        ctx.out(tcid, t);

        /* Step 6 - Create Tuple Template */
        LogicTuple tt = new LogicTuple("p", new Var("X"));

        /* Step 7 - Retreive the Tuple matching the Tuple Template */
        LogicTuple rt = ctx.in(tcid, tt);

        /* Step 8 - Print the Retreived Tuple */
        System.out.println(rt);
```

```
        /* Step 9 - Exit ACC */
        ctx.exit();

    }
}
```

Listing 4.4 – Hello World

Step 1. On crée un `AgentId` ayant pour identifiant "*user*".

Step 2. L'*agent coordination context* par défaut (`ctx`) est obtenu. L'invocation de la méthode statique `Tucson.getContext(aid)` renvoie un objet de type `AgentContextStub`. C'est cette classe qui implémente l'interface `TucsonContext` définie précédemment. L'utilisation de la classe `TucsonContextSynch` va permettre de travailler avec un *agent coordination context* de manière synchronisée, c'est-à-dire en attendant l'achèvement (la complétude) de chaque invocation avant de poursuivre.

Step 3. On crée `TupleCentreId` ayant pour nom "*test*".

Step 4. Un *logic tuple* `p` est créé et prend comme argument la chaîne de caractère "*hello world*".

Step 5. On invoque la méthode `out`, c'est-à-dire qu'on insère un *logic tuple* dans le *tuple centre* via l'*agent coordination context* obtenu par l'*agent*.

Step 6. On crée un *tuple template* `p` qui prend une variable en argument (notée par "X"[15]).

Step 7. On invoque la méthode `in`, c'est-à-dire qu'on attend (opération bloquante) qu'un *logic tuple* correspondant au *tuple template* soit inséré dans le *tuple centre*. Ici, vu que le *logic tuple* a été inséré quelques lignes avant, l'opération se déroule assez vite.

Step 8. On affiche dans la console le *logic tuple* trouvé dans le *tuple centre* correspondant au *tuple template*. La chaîne de caractère affichée est :

```
p('hello world')
```

Step 9. On libère l'*agent coordination context* obtenu au début.

L'outil **TuCSoN** *Inspector* fourni avec l'API permet d'observer n'importe quel *tuple centre* dans l'infrastructure **TuCSoN** déployée. Cet outil est lancé de la même manière que le noeud ou le CLI est vu comme un *agent* normal (ayant pour identifiant "*inspector*") dans l'infrastructure.

```
java alice.tucson.tools.Inspector
```

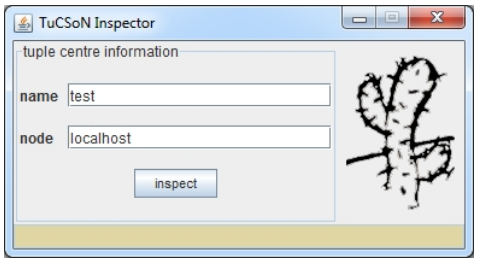

FIGURE 4.12 – Ecran d'accueil de l'Inspector

Après avoir mentionné le nom du tuple centre et l'adresse IP du noeud, on est dirigé vers l'écran présenté à la figure 4.13

Ici, plusieurs choix s'offrent à nous. On peut observer les *tuples* présents dans le *tuple centre* (bouton **tuples**) et avoir une vue des invocations en attente (bouton **pending**), qui sont principalement des requêtes bloquantes de type `in` et `rd`. De même, on peut vérifier l'ensemble des réactions

15. On remarque ici l'utilisation d'une majuscule pour indiquer le fait que "X" soit une variable.

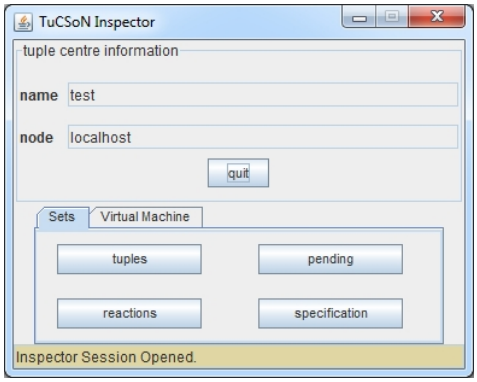

FIGURE 4.13 – Ecran d'accueil de l'Inspector (tuple centre et noeud mentionnés)

(bouton **specification**) qui ont été programmées dans le *tuple centre* et voir leurs effets (bouton **reactions**) lorsque des invocations de primitives de coordination ont déclenché des réactions [16].

Dans l'exemple "*Hello World*", en s'arrêtant avant l'étape 7, on peut voir le *logic tuple* présent dans le *tuple centre* (figure 4.14)

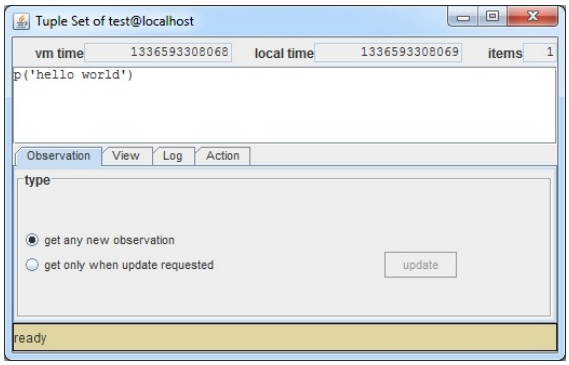

FIGURE 4.14 – Contenu du tuple centre

L'invocation de la primitive `in` à l'étape 7 se déroule trop rapidement que pour être observée. Toutefois, sans placer de *tuple* dans le *tuple centre*, on peut "singer" ce qu'il y a eu dans l'écran des requêtes en attente (figure 4.15).

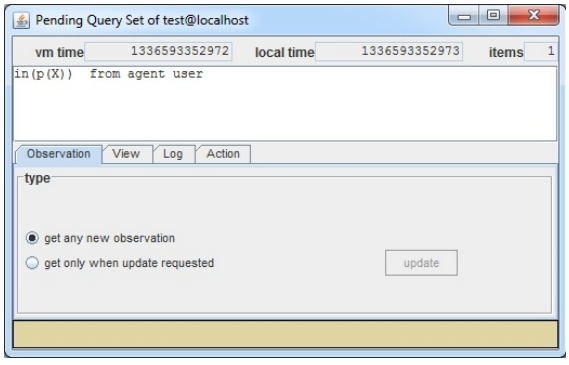

FIGURE 4.15 – Requête en attente dans le tuple centre

16. Un exemple concernant les réactions sera présenté plus loin dans le document.

4.6 Exploitation

TuCSoN est axé sur le design, le développement et la gestion temps réel des tâches "sociales [17]", c'est-à-dire la partie coordination du système. La conception de ces éléments implique essentiellement :

- *Tuple centre cardinality and location.* Etablir combien et quels *tuple centres* utiliser, ainsi que leur localisation en termes de noeuds. Différentes stratégies peuvent être adoptées, allant d'envisager un simple *tuple centre* pour permettre et gouverner n'importe quelle interaction d'*agent*, à envisager un *tuple centre* pour chaque *agent* du système, agissant comme sa boîte à message privée. Typiquement, un *tuple centre* est nécessaire pour chaque tâche de coordination tandis que les tâches indépendantes de coordination seront supportées par des *tuple centres* distincts. Comme règle générale, on peut introduire un *tuple centre* lorsqu'il y a des dépendances (interactions) parmi les *agents* que l'on veut capturer et gérer ; en d'autres termes, on peut introduire un *tuple centre* lorsqu'on a besoin d'un artefact de coordination pour suporter un ensemble d'*agents* dans l'accomplissement d'un but partagé, afin de supporter leur travail coopératif.
- *Tuple-based protocols.* Etablir le format des *tuples* utilisés et échangés, ainsi que les protocoles apparentés, spécifiant la séquence d'insertion/retrait/lecture des *tuples* dans et à partir des *tuple centres*.
- *Tuple centre coordinating behaviour.* Programmer le comportement des *tuple centres* dans le but de réaliser les politiques/lois de coordination qui caractérisent les tâches "sociales".

4.6.1 Modèles de communication

4.6.1.1 Blackboard

Un système *blackboard* est une application où une base de connaissances commune (le tableau noir, tel qu'on se le représente dans une salle de classe) est mise à jour de manière itérative par divers groupes de sources de connaissances spécialisées, commençant par une spécification du problème et finissant par une solution. Chaque source de connaissances met à jour le *blackboard* avec une solution partielle, lorsque ses contraintes internes correspondent à l'état du *blackboard*. De cette manière, les spécialistes travaillent ensemble pour résoudre le problème. A l'origine, le modèle *blackboard* a été conçu comme un moyen de manipuler des problèmes complexes, mal définis, où la solution est la somme de ses éléments constituant.

Il est évident que la communication basée sur le *blackboard* est supportée naturellement par **TuCSoN**. Les *agents* communiquent en écrivant des informations sur un *blackboard* (ce qui revient à insérer des *tuples* dans un *tuple centre*) et lisent/nettoyent de telles informations (ce qui revient à lire ou à enlever des *tuples* des *tuple centres*), en exploitant le *pattern matching*. Cette forme de communication, qui est appelée la *communication générative* dans le cas des *tuple centres*, présente un certain nombre de propriétés :

- *space uncoupling* - la communication est possible même si les participants ne savent pas où ils se trouvent réellement.
- *time uncoupling* - la communication est possible même si les participants ne sont pas simultanément dans le même contexte d'interaction, voire en vie en même temps.
- *identity uncoupling* - la communication est possible même si les participants ne savent pas qui ils sont réellement.

Ces propriétés rendent cette approche adéquate en particulier pour des systèmes ouverts et dynamiques, avec un ensemble dynamique de participants, dont l'identité peut être inconnue durant les activités de communication.

17. Dans des systèmes complexes, certaines tâches ont inévitablement besoin d'accéder à des ressources distribuées et partagées : elles endossent donc une nature sociale.

4.6.1.2 Message

TuCSoN peut être facilement adopté pour réaliser une communication classique d'*agents* basée sur les messages. Dans l'exemple qui suit, un simple design est adopté, mais il peut facilement être étendu suivant les besoins de l'application.

- Un *tuple centre* `msgbox` est choisi en tant que boîte à message privée (ou *blackboard*), où les messages sont placés et enlevés.
- Un *tuple* `msg(Dest, Msg)` est utilisé pour représenter un message : ***Dest*** représente l'identifiant (logic/virtuel) de l'*agent* récepteur du message ; ***Msg*** représente le message.
- Le protocole :
  - `out(msg(Dest, Msg)` pour envoyer un message ***Msg*** à un *agent* ***Dest***.
  - `in(msg(Dest, ?Msg)` pour recevoir un message (publié pour l'*agent* ***Dest***, cible du message).
  - `inp(msg(Dest, ?Msg)` pour recevoir un message, sans bloquer dans le cas où aucun message n'est disponible.

Le code repri dans le listing 4.5 montre une implémentation basique de la classe de l'expéditeur du message.

```
import alice.tucson.api.*;
import alice.logictuple.*;

public class TestSend {
    public static void main(String[] args) throws Exception {

        /* Tuple Centre used as a Message Box */
        TupleCentreId tid = new TupleCentreId("msgbox@localhost");

        /* Receiver of the message */
        Value dest = new Value("getter");

        /* Content of the message - a simple String */
        Value msg = new Value("hello from sender!");

        /* Get context */
        TucsonContextSynch ctx = new TucsonContextSynch(Tucson.getContext(
              new AgentId("sender")));

        /* Send the message */
        ctx.out(tid, new LogicTuple("msg",dest,msg));

        /* Exit Context */
        ctx.exit();
    }
}
```

Listing 4.5 – TestSend Class

Un *tuple centre* `msgbox` est convenu entre les deux parties. Dans l'exemple, ce *tuple centre* est situé dans un noeud local (`@localhost`), mais il peut très bien être situé dans un noeud distant (l'adresse IP doit alors être encodée entre apostrophes, `@'192.168.1.5'`).

Le nom du destinataire est `getter`, tandis que le nom de l'expéditeur est `sender`. Ces noms sont aussi utilisés en tant qu'identifiant des *agents* respectifs. Le message est envoyé via un *tuple* dont le nom est `msg`.

Le code pour le destinataire (listing 4.6) peut donc être représenté différemment selon que l'on utilise la primitive de communication bloquante (`in`) ou non bloquante (`inp`). Afin de n'avoir qu'une seule classe pour le destinataire, on utilisera une varibale booléenne qui permettra de choisir un mode ou l'autre.

```
import alice.tucson.api.*;
import alice.logictuple.*;

public class TestReceive {

    public static void main(String[] args) throws Exception {

        /* Boolean - blocking or not blocking */
        Boolean blocking = true;

        /* Tuple Centre used as a Message Box */
        TupleCentreId tid = new TupleCentreId("msgbox@localhost");

        /* Identifier of the Receiver */
        Value myid = new Value("getter");

        /* Get Context */
        TucsonContextSynch ctx = new TucsonContextSynch(Tucson.getContext(
              new AgentId(myid.toString())));

        /* Get message */
        LogicTuple msg;
        if(blocking)
            msg = ctx.in(tid, new LogicTuple("msg", myid, new Var("Msg")));
        else
            msg = ctx.inp(tid, new LogicTuple("msg", myid, new Var("Msg")));

        if(msg == null)
            System.out.println("No message for " + myid);
        else
            System.out.println("New message for " + myid + " : " + msg.getArg(1));

        /* Exit Context */
        ctx.exit();
    }
}
```

Listing 4.6 – TestReceive Class

Dans le cas où la variable `blocking` est à `false`, on invoque la méthode `inp` non bloquante. Si il n'y avait pas de messages dans le *tuple centre* `msgbox`, la variable `msg` prend la valeur `null`. La console affiche alors :

```
No message for getter
```

Dans le cas où la variable `blocking` est à `true`, on invoque la méthode `in` bloquante. La variable `msg` ne prendra donc jamais la valeur `null` puisque l'*agent* attendra infiniment qu'un message arrive dans le *tuple centre*.

Dans ces deux cas, lorsqu'un message arrive au destinataire, la console affiche :

```
New message for getter : 'hello from sender!'
```

Dans cet exemple, on envoie comme message une simple chaîne de caractères : en fait, il est possible d'échanger n'importe quelle sorte d'informations complexes ou structurées codées comme des *logic tuples* (incluant des images, des objets **Java** sérialisés, ... encodés en tant que paquets de bytes, encapsulés dans des chaînes de caractères).

4.6.1.3 Client-Serveur

Une communication client-serveur - que l'on trouve dans les architectures *Remote Procedure Call* ou *Service-Oriented* (telles que les *Web Services*) - peut être modélisée avec un simple protocole basé sur les *tuple centres*. On considère les points suivant :

- Un *tuple centre* est adopté comme médiateur entre les clients et les fournisseurs de services.
- Un *tuple* `request(Who, What)` est utilisé pour représenter une requête de service, émise par le client ***Who***, décrite par ***What***. Un *tuple* `result(Who, What, Request)` est utilisé pour représenter le résultat du service, émis par un fournisseur de service répondant à la requête précédente émise par le client ***Who***, décrite par ***What***.
- Le protocole :
  - Les clients demandent un service en insérant un *tuple* `request` dans le *tuple centre*, avec
    `out(request(ClientID, What))`
    ils collectent après les résultats en récupérant le *tuple* avec
    `in(result(ClientID, What, ?Result))`
  - Les fournisseurs de service acceptent une requête en enlevant un *tuple* `request` avec
    `in(request(?ClientID, What, ?Result))`
    et fournissent les résultats en insérant le *tuple* correspondant
    `out(result(ClientID, What, Result))`

Il est évident que pour des applications réelles, des protocoles plus robustes et mieux articulés peuvent être adoptés.

L'exemple qui suit implémente un simple fournisseur de service capable de calculer la somme de deux éléments (nombres entiers) et de renvoyer le résultat au client qui a fait la requête. L'implémentation du fournisseur de services est gérée avec la classe `ServiceProviderAgent` (listing 4.7) qui hérite de la classe `Thread`. La méthode `run()` est en charge de la gestion des requêtes entrantes et du calcul du résultat de chaque requête.

```
import alice.tucson.api.*;
import alice.logictuple.*;

public class ServiceProviderAgent extends Thread {

    private TupleCentreId tid;

    public ServiceProviderAgent(TupleCentreId tid) {
        this.tid = tid;
    }

    @Override
    public void run() {
        try {

            /* Get Context for ServiceProvider Agent */
            TucsonContextSynch ctx = new TucsonContextSynch(Tucson.getContext(
                                                    new AgentId("SPAgent")));
            System.out.println("ServiceProviderAgent started.");

            while(true) {

                /* Wait for request */
                LogicTuple req = ctx.in(tid,
                        new LogicTuple("request", new Var("Who"),
                            new Value("sum", new Var("X"), new Var("Y"))));
                System.out.println("A new request arrived : " + req);

                try {

                    /* Calculate and send result */
                    TupleArgument who = req.getArg(0);
                    int x = req.getArg(1).getArg(0).intValue();
                    int y = req.getArg(1).getArg(1).intValue();
```

```
                    int res = x + y;
                    ctx.out(tid, new LogicTuple("result",
                                 who,
                                 req.getArg(1),
                                 new Value(res)));

                } catch(Exception ex) {
                    ex.printStackTrace();
                    System.err.println("Error in processing the request.");
                }
            }

        } catch(Exception ex) {
            System.err.println("Problems in accessing TuCSoN.");
        }
    }

}
```

Listing 4.7 – ServiceProviderAgent Class

La méthode `getArg(int index)` exécutée sur le *tuple* `req` permet d'obtenir la variable à l'index spécifié. La variable `Who` est située à l'index 0 tandis que le *tuple* `sum` se trouve à l'index 1. Ce *tuple* est lui-même composé de deux variables auxquelles on peut accéder en invoquant aussi la méthode `getArg(int index)`.

On écrit une méthode `main` (listing 4.8) qui lance le thread lié à l'*agent* fournisseur de services.

```
import alice.tucson.api.*;
import alice.logictuple.*;

public class ServiceProvider {

    /* Main method - starts ServiceProviderAgent thread */
    public static void main(String[] args) throws Exception {
        new ServiceProviderAgent(new TupleCentreId("service_center")).start();
    }
}
```

Listing 4.8 – ServiceProvider Class

Le client envoie sa requête sur le *tuple centre* `service_center` qui est le *tuple centre* sur lequel le fournisseur de services traite les requêtes. L'implémentation de l'*agent* demandeur est décrite dans le listing 4.9

```
import alice.tucson.api.*;
import alice.logictuple.*;

public class ServiceRequester {

    public static void main(String[] args) throws Exception {

        /* Get Context for ServiceRequester Agent */
        TucsonContextSynch ctx = new TucsonContextSynch(Tucson.getContext(
                                                        new AgentId("SRAgent")));
        TupleCentreId tid = new TupleCentreId("service_center");

        Value who = new Value("client0");
        Value x = new Value(54);
        Value y = new Value(17);

        System.out.println("Agent " + who
        + " requesting the sum of " + x + " and " + y + " ...");
```

```
        /* Ask service */
        Value service = new Value("sum", x, y);
        ctx.out(tid, new LogicTuple("request", who, service));

        /* Wait for result */
        LogicTuple res = ctx.in(tid,
                    new LogicTuple("result", who, service, new Var("R")));
        int result = res.getVarValue("R").intValue();
        System.out.println("The result is " + result);

        /* Exit Context */
        ctx.exit();
    }
}
```

Listing 4.9 – ServiceRequester Class

Le tableau (4.6) liste ce que le fournisseur de services et le client ont successivement affiché dans leur console respective.

Agent ServiceProvider	**Agent ServiceRequester**
`ServiceProviderAgent started.`	`Agent client0 requesting the sum of 54 and 17 ...`
`A new request arrived : request(client0,sum(54,17))`	`The result is 71`

TABLE 4.6 – ServiceProvider & ServiceRequester

4.6.2 Persistance

Dans la version actuelle de **TuCSoN** (1.4.5), un support basique de la persistance a été implémenté. Ce service permet de rendre dynamiquement persistants certains ensembles spécifiques de *tuple centres* : lorsqu'un *tuple centre* est persistant, son contenu est préservé quand le noeud s'arrête, soit à cause d'une extinction manuelle, soit à cause d'une défaillance. Lorsque le noeud est relancé, les *tuple centres* pour lesquels la persistance était activée sont automatiquement restaurés.

Le service de persistance peut être exploité en interagissant de façon appropriée avec le *tuple centre* `$ORG` du noeud. Le protocole de ce service est le suivant :

- Pour activer la persistance :

  ```
  '$ORG' @ Node ? out(cmd(enable_persistency(TCName)))
  ```

  Node est le noeud **TuCSoN** en question ; ***TCName*** est un *template* définissant l'ensemble des *tuple centres* à rendre persistant. Ce *template* peut être directement le nom d'un *tuple centre*, aussi bien qu'un ensemble de variables définissant un ensemble de noms. L'effet de l'invocation est de rendre persistant chaque *tuple centre* dont le nom correspond au *template* spécifié.

  Afin de vérifier si la commande est réussie, on peut envoyer la commande suivante :

  ```
  '$ORG' @ Node ? out(cmd(enable_persistency(TCName)), Result)
  ```

Si cette commande a réussi, ***Result*** est unifié avec l'atome `ok` ; sinon, `unknown` est renvoyé si la commande est inconnue, `failed` si la commance a échoué.

- Pour désactiver la persistance :

  ```
  '$ORG' @ Node ? out(cmd(disable_persistency(TCName)))
  ```

  Node est le noeud **TuCSoN** en question ; ***TCName*** est un *template* définissant l'ensemble des *tuple centres* (rendus persistants précédemment) à rendre non persisants.

  Comme dans le cas précédent, on peut envoyer la commande suivante afin de vérifier si elle est réussie :

  ```
  '$ORG' @ Node ? out(cmd(disable_persistency(TCName)), Result)
  ```

  Si la commande a réussi, ***Result*** est unifié avec l'atome `ok` ; sinon, `unknown` est renvoyé si la commande est inconnue, `failed` si la commance a échoué.

- Pour vérifier si un *tuple centre* est persistant :

  ```
  '$ORG' @ Node ? rdp(is_persistent(TCName))
  ```

  Node est le noeud **TuCSoN** en question ; ***TCName*** est le nom du *tuple centre* à vérifier. La requête se termine avec succès si le *tuple centre* ***TCName*** est persistant.

La stratégie (basique) adoptée dans cette version consiste à créer un fichier texte pour chaque *tuple centre* rendu persistant, dont le nom équivaut à `tc_` concaténé avec le nom du *tuple centre*. Les fichiers sont enregistrés dans un sous répertoire nommé `persistent` où le noeud **TuCSoN** est lancé. Chaque fichier texte contient une capture instantanée du contenu du *tuple centre* au moment où la persistance a été activée, ainsi que la liste de toutes les opérations exécutées sur l'ensemble des *tuples* qui ont pu changer son contenu. Lorsqu'un noeud est lancé, tous les fichiers textes contenus dans le dossier `persistent` sont inspectés et les *tuple centres* dont il est question sont restaurés.

4.6.3 Gestion du temps

Dans la plupart des scénarios d'application caractérisés par un haut degré de dynamisme et de transparence, les tâches de coordination ont besoin d'être dépendantes du temps. D'une part, il est très utile de spécifier (et puis de mettre en oeuvre) des niveaux donnés d'existence et de qualité de services, comme par exemple, exiger que les *agents* interagissent avec des artefacts de coordination à une fréquence minimum/maximum. D'autre part, les propriétés temporelles sont aussi des aspects fondamentaux concernant l'interception des violations dans les contrats des artefacts d'*agents* : un *agent* peut être amené à fournir un service dans un temps limite donné, ou il peut exiger la même contrainte aux artefacts. En conséquence, la version 1.4.0 de **ReSpecT** a été étendue pour inclure les capacités de gestion des événements temporels, afin de supporter la définition et l'adoption de politiques de coordination *time-aware*. L'idée de base est d'exploiter la programmabilité du médium de coordination étendu avec un *framework* temporel afin d'obtenir la capacité de modéliser n'importe quel gabarit de coordination basé sur le temps, réalisé directement en spécifiant un comportement adéquat de l'artefact de coordination.

Dans un premier temps, le modèle **ReSpecT** est étendu avec une notion de temps actuel `Tc` (*current time*) du *tuple centre* : chaque *tuple centre* a sa propre horloge, qui définit l'écoulement du temps (l'unité étant la milliseconde). En fait, le temps du *tuple centre* est un temps physique, mais sa valeur est considérée comme étant constante durant l'exécution d'une réaction individuelle : en d'autres mots, on considère que `Tc` se réfère au temps lorsque la réaction a commencé son exécution.

Ce choix est cohérent avec la philosophie **ReSpecT** concernant les réactions, qui sont destinées à être exécutées atomiquement (dans le cas des réactions réussies).

Afin d'obtenir le temps actuel `Tc` dans les programmes **ReSpecT**, une nouvelle primitive est introduite :

```
current_time(?Tc)
```

Cette primitive (ou prédicat) est réussie si ***Tc*** (une variable) s'unifie avec le temps actuel `Tc` du *tuple centre*. Considérons par exemple la réaction suivante :

```
reaction(in(p(X)), (
   current_time(Tc),
   out_r(request_log(Tc,p(X)))
   )).
```

Cette réaction insère un nouveau *tuple* avec une information temporelle à chaque fois qu'une requête de retrait d'un *tuple* `p(X)` est exécutée, réalisant ainsi un *log* de ces requêtes.

Le modèle **ReSpecT** est ensuite étendu avec la notion de *trap event* (ou simplement *trap*) qui est un événement généré lorsqu'un *tuple centre* atteint un point temporel spécifique. Une *trap* survient à cause d'une *trap source*, caractérisée par un identifiant unique ***ID***, un temps ***Te*** et un *description tuple* ***Td***. Le langage est étendu avec la possibilité de générer et manipuler des *trap events* et des *trap sources*. Plus particulièrement, on introduit les caractéristiques suivantes :

- intérieurement dans le *tuple centre*, une loi de coordination (c'est-à-dire, un ou plusieurs *tuples* de spécification de réactions) peut installer une *trap source*, qui amène une *trap* à survenir à un temps spécifique. Par exemple, on peut vouloir générer une *trap* décrite par le *tuple* `expired(T)` un certain interval ***LeaseTime*** après l'insertion d'un *tuple* `leased(T)` ;
- le *tuple centre* réagit à une *trap* de manière analogue aux événements de communication, au moyen de *tuples* de spécification de réactions propres. Dans le cas précédemment décrit, on veut que le *tuple* ***T*** soit enlevé lorsqu'une *trap* décrite par `expired(T)` survient.

Afin de supporter l'installation de générateur de *trap*, le langage est étendu avec deux nouvelles primitives (la notation **Prolog** est adoptée pour décrire la modalité des arguments : `+` est utilisé pour spécifier un argument d'entrée, `-` pour un argument de sortie, `?` pour un argument d'entrée/sortie, `@` pour un argument d'entrée qui doit être complètement instancié) :

```
new_trap(-ID,@Te,+Td)
```

```
kill_trap(@ID)
```

La première primitive est résussie si ***Te*** est un entier plus grand ou égal à zéro. Son effet est d'installer une nouvelle *trap source* - avec l'identifiant ***ID*** - qui entre dans une file de sources installées. Lorsque le temps `Tc` du *tuple centre* sera égal ou plus grand au temps actuel plus ***Te***, un *trap event* décrit par le *tuple* ***Td*** sera généré et inséré dans la file des *trap events* déclenchés, alors que sa source sera désinstallée (c'est-à-dire, enlevée de la file). Par exemple, La réaction suivante :

```
reaction(out(leased(T,LeaseTime)), (
   new_trap(_,LeaseTime,expired(T))
   )).
```

est déclenchée lorsqu'un *tuple* correspondant à `leased(T,LeaseTime)` est inséré. Cela installe une nouvelle *trap source* qui générera une *trap* décrite par le *tuple* `expired(T)` après ***LeaseTime*** unités de temps. La primitive `kill_trap` est, quant à elle, utilisée pour désinstaller une source en donnant son identifiant : une telle primitive échouera si aucune source installée n'est caractérisée par l'identifiant fourni.

Le langage a aussi été étendu avec la possibilité d'écrire des réactions déclenchées par l'occurence de *trap events*. Les modèles syntaxiques et sémantiques de réactions de type *trap* sont analogues aux réactions d'événements de communication.

```
reaction(trap(Tuple),Body).
```

Body spécifie l'ensemble des actions devant être exécutées lorsqu'une *trap* avec un *description tuple* correspondant au *template* ***Tuple*** survient.

```
reaction(trap(expired(T)),(in_r(T))).
```

Dans l'exemple précédemment décrit, lorsqu'une *trap* décrite par un *tuple* correspondant au *template* `expired(T)` survient, le *tuple* spécifié par ***T*** est enlevé de l'ensemble des *tuples*. Il est à noter que si le *tuple* n'est pas présent, `in_r` échoue causant l'échec de l'entièreté de la réaction. Cependant, vu que le *trap event* survient, la *trap source* est effacée.

Les *trap events* sont "entendus" un par un dès que le *tuple centre* n'est plus en train d'exécuter une réaction. Lorsqu'un *trap event* est entendu, il est tout d'abord enlevé de la file des *trap events*, l'ensemble des réactions qu'il déclenche est déterminé - par correspondance de l'en-tête de réaction avec le *description tuple* du *trap* - et sont ensuite exécutées séquentiellement.

Un aspect important de la sémantique de cette extension concerne la priorité des réactions lancées par des événements de communication (exécution standard) par rapport à celle des *trap events* (exécution *trap*). Le modèle et l'implémentation décrits ici figurent une plus haute priorité pour les réactions lancées par des *trap events*. Cela signifie que si, durant les exécutions standards d'une chaine de réactions, il survient un *trap event*, la chaine est cassée, et les réactions lancées par la *trap* sont exécutées. Il est important de noter que les réactions individuelles sont toujours atomiques (non interruptibles) telles que dans le modèle basique de **ReSpecT** : les *trap events* dans la file de *traps* sont entendus (et les réactions liées sont exécutées) après la complétion de n'importe quelle éventuelle réaction en exécution. Les chaines de réactions peuvent être scindées, par les réactions individuelles, ce qui est fondamental afin de préserver les propriétés sémantiques du modèle **ReSpecT**. De manière analogue, les réactions déclenchées par un *trap event* sont atomiques, elles ne peuvent pas être interrompues ou suspendues : en d'autres mots, les gestionnaires de *traps* ne sont pas interruptibles et ils ne peuvent pas être imbriqués.

Néanmoins, il est important de mentionner que d'autres sémantiques sont possibles et intéressantes. En donnant une priorité plus élevée à l'exécution standard, on s'assure que les *traps* n'interfèrent jamais dans leur exécution. En échange d'une meilleure isolation du code obtenu, dans ce cas, on ne peut plus garantir les mêmes contraintes de temps : les exécutions de *traps* doivent attendre que les exécutions standards soient terminées. Il est à noter que de tels aspects sont principalement orthogonaux à l'application réelle des lois temporelles de coordination. De plus, une généralisation directe du modèle peut être obtenue en spécifiant un niveau de priorité d'une *trap* (plus élevé, moins élevé, ou identique aux événements de communication externes) au moment où la source est installée. Cependant, cette caractéristique est toujours en cours de développement et ne sera donc pas décrite/utilisée dans ce document.

4.6.3.1 Tuples en leasing

Dans l'exemple qui suit, on modélise la notion de *leasing*. Les *tuples* peuvent être insérés dans le *tuple set* en spécifiant un temps de bail (*lease time*), c'est-à-dire le temps maximum qu'ils peuvent résider dans le *tuple centre* avant leur enlèvement automatique.

```
1  reaction( out(leased(Time,Tuple)), (
      new_trap(Id,Time,lease_expired(Time,Tuple)),
      in_r(leased(Time,Tuple)),
      out_r(outl(ID,Time,Tuple)) )).

2  reaction( rd(Tuple), (
      pre,
      rd_r(outl(ID,_,Tuple)),
      out_r(Tuple) )).

3  reaction( rd(Tuple), (
      post,
      rd_r(outl(ID,_,Tuple)),
      in_r(Tuple) )).

4  reaction( in(Tuple), (
      pre,
      in_r(outl(ID,_,Tuple)),
      out_r(Tuple),
      kill_trap(ID) )).

5  reaction( trap(lease_expired(Time,Tuple)), (
      in_r(outl(ID,Time,Tuple)) )).
```

TABLE 4.7 – Tuples en leasing

Un *agent* insère un *tuple* avec un *lease time* en invoquant une commande :

```
out(leased(@Time,@Tuple)
```

Le listing 4.7 montre les spécifications **ReSpecT** programmant le *tuple centre* avec le comportement de *leasing* voulu. Lorsqu'un *tuple* est inséré avec un *lease time* dans le *tuple centre*, une *trap source* est installée pour générer une *trap* lorsque le temps du *tuple centre* atteint le *lease time* spécifié (reaction 1). Aussi, un *tuple* `outl` est inséré dans le *tuple space* avec les informations de la *trap source* et du *leased tuple* (il est à noter que le (*flat*) *tuple* avec le *lease time* n'est pas directement présent dans le *tuple set*). Alors, pour chaque requête `rd` invoquée avec un *template* correspondant un *leased tuple*, un (*flat*) *tuple* satisfaisant la requête est d'abord inséré dans le *tuple set* (réaction 2), et après enlevé lorsque la requête `rd` a été satisfaite (réaction 3). Une requête `in` cause plutôt l'enlèvement du *lease tuple* et de la *trap source* (réaction 4). Finallement, lorsqu'un *trap event* survient (signifiant que le *lease time* du *tuple* a expiré), le *tuple* `outl` portant l'information à propos de la présence du *leased tuple* est enlevé (réaction 5).

4.6.3.2 Exemple d'implémentation d'un log

L'exemple qui suit implémente en **Java** un système de *log* tel que décrit précédemment. L'outil **TuCSoN** *Inspector* fourni avec l'API va permettre de visualiser les réactions déclenchées.

On crée dans un premier temps une classe (listing 4.10) qui va remplir un *tuple centre* `letters` avec les vingt-six lettres de l'alphabet. Cette classe va ensuite programmer la spécification en charge du système de *log*. Afin de s'assurer du fonctionnement du système, on affichera dans la console chaque ligne de *log*.

```
import alice.tucson.api.*;
import alice.logictuple.*;

public class LogSystem {

    public static void main(String[] args) throws Exception {

        /* Get 'logger' agent's context */
        TucsonContextSynch ctx = new TucsonContextSynch(Tucson.getContext(
                                                        new AgentId("logger")));

        /* Create Tuple Centre ID */
        TupleCentreId tcid = new TupleCentreId("letters");

        /* Fill Tuple Centre for test purpose */
        String letters = "abcdefghijklmnopqrstuvwxyz";
        for(int i = 0; i < letters.length(); i++)
            ctx.out(tcid, new LogicTuple("letter",
                              new Value(String.valueOf(letters.charAt(i)))));

        /* Set specification */
        String reaction = "reaction(inp(letter(X)),("
                + "post,"
                + "current_time(Tc),"
                + "out_r(request_log(Tc,letter(X)))));";

        ctx.setSpec(tcid, reaction);

        /* Print each 'request_log' tuples */
        LogicTuple template = new LogicTuple("request_log",
                                        new Var("X"), new Var("Y"));
        LogicTuple log;

        while(true) {
            log = ctx.in(tcid, template);
            System.out.println("New log : " + log);
        }

    }
}
```

Listing 4.10 – LogSystem Class

La spécification se présente comme suit :

```
reaction(inp(letter(X), (
   post,
   current_time(Tc),
   out_r(request_log(Tc,letter(X))) )).
```

A chaque fois qu'un *agent* voudra enlever un *tuple* (une lettre) du *tuple centre* `letters`, on enregistrera un nouveau *tuple* `request_log` reprenant le *tuple* enlevé et l'information temporelle lors de l'exécution de la réaction. On utilise ici le prédicat `post` pour ne déclencher la réaction qu'une seule fois, lors de la réussite de l'exécution du prédicat `inp`.

Le tuple centre contient bien vingt-six *tuples* représentant les lettres de l'alphabet, tel que montré dans la figure 4.16.

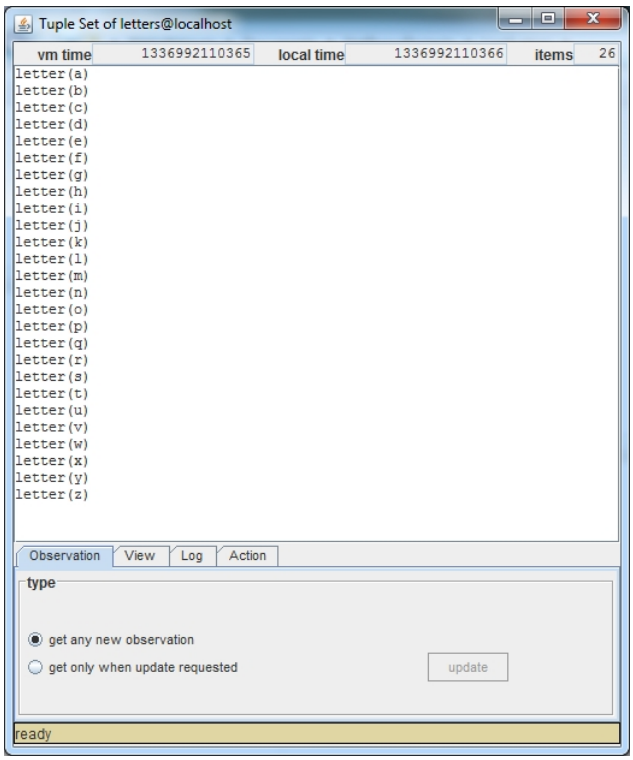

FIGURE 4.16 – Contenu du tuple centre

La figure 4.17 montre la requête en attente de l'*agent* `'logger'`, puisqu'il est bloqué avec le prédicat `in` le temps qu'un *tuple* `request_log` arrive dans le *tuple centre*.

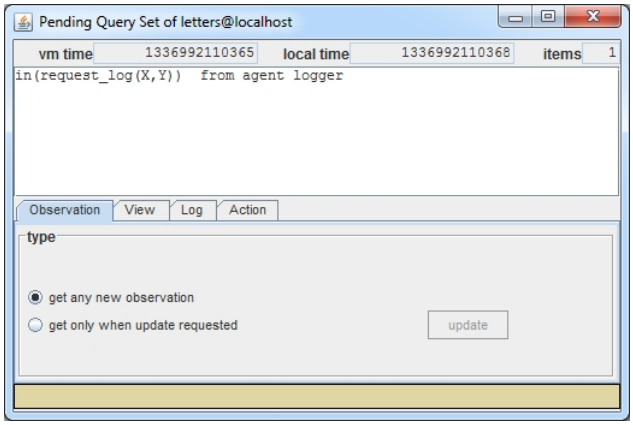

FIGURE 4.17 – Requête en attente

La spécification du comportement du *tuple centre* `letters` est montré à la figure 4.18.

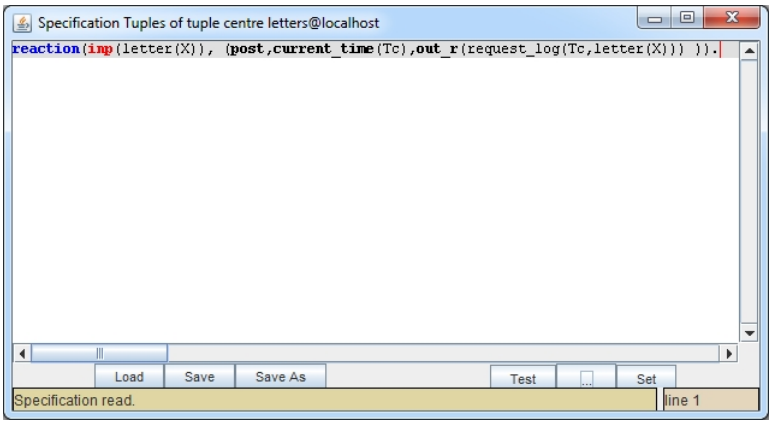

FIGURE 4.18 – Spécification

L'implémentation de l'*agent* en charge de retirer les *tuples* `lettre` du *tuple centre* est montré dans le listing 4.11. Cet *agent* choisit une lettre de manière aléatoire. On utilise le prédicat non bloquant `inp` pour empêcher le système d'attendre indéfiniment au cas où le nombre aléatoire prendrait plusieurs fois la même valeur.

```
import alice.tucson.api.*;
import alice.logictuple.*;
import java.util.Random;

public class RequesterAgent {

    public static void main(String[] args) throws Exception {

        /* Get 'requester' agent's context */
        TucsonContextSynch ctx = new TucsonContextSynch(Tucson.getContext(
                                 new AgentId("requester")));

        /* Create Tuple Centre ID */
        TupleCentreId tcid = new TupleCentreId("letters");

        /* Choose a letter to retreive */
        String letters = "abcdefghijklmnopqrstuvwxyz";
        Random rand = new Random();
        char letter = letters.charAt(rand.nextInt(letters.length()));

        ctx.inp(tcid, new LogicTuple("letter", new Value(String.valueOf(letter))));

        /* Exit context */
        ctx.exit();
    }
}
```

Listing 4.11 – RequesterAgent Class

En démarrant cet *agent*, on remarque qu'un *tuple* `lettre` aléatoire est bien enlevé du *tuple centre*. Le déclenchement de la réaction peut aussi être observé grâce à l'*Inspector* (figure 4.19).

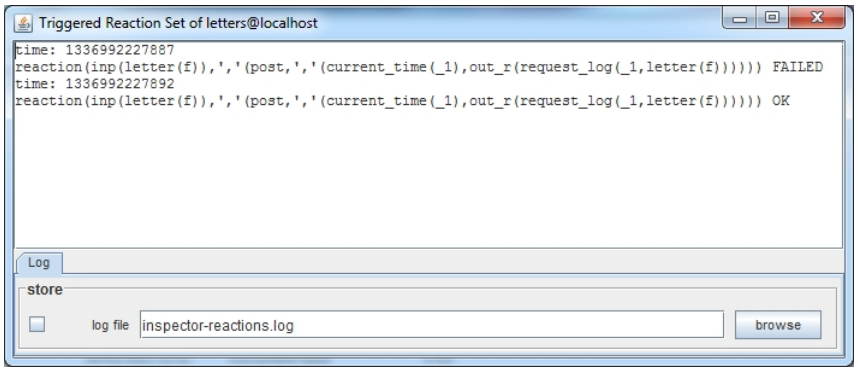

FIGURE 4.19 – Réactions

Tel que décrit précédemment, la réaction est déclenchée deux fois ; une fois dans le *pre stage* et une fois dans le *post stage*. Dans le *pre stage*, la réaction échoue [FAILED], tandis qu'elle réussit dans le *post stage* [OK], vu que l'on avait volontairement indiqué le prédicat `post` dans la spécification du *tuple centre*.

Dans la console de l'*agent* `'logger'`, on peut alors lire :

```
New log : request_log(205121,letter(f))
```

Ce qui correspond bien au retrait du *tuple* `lettre(f)` du *tuple centre* tel que montré dans l'écran des réactions de l'*Inspector*.

4.7 Conclusion

Les notions de **TuCSon** présentées dans ce document permettent de se rendre compte de la portée que peut avoir son implémentation dans un système applicatif. L'infrastructure de coordination scinde de manière hiérarchique l'espace de coordination en entités (les noeuds) et sous-entités (les *tuple centres*), donnant ainsi lieu à une séparation modulaire, conceptuelle, permettant de calquer plus aisément une application construite au-dessus de l'infrastructure de **TuCSoN**. Néanmoins, cette scission implique intrinsèquement la compréhension et l'étude complète de l'application avant son implémentation.

Le concept de programmabilité dynamique introduit à l'aide du langage **ReSpecT** déplace une partie de l'intelligence du système directement dans les média de coordination (les *tuple centres*), allégeant ainsi l'applicatif. On se retrouve toujours dans une infrastructure partagée comme pour les *blackboards* ou les *tuple spaces*, mais dont l'organisation et le routage de l'information (les *tuples*) sont régis par des lois de coordination encapsulées dans les média de coordination. Ce flux d'information est donc rendu transparent pour la partie applicative du système à implémenter.

L'idée générale présentée dans l'approche organisationnelle du modèle **TuCSoN** est d'impliquer davantage les notions de rôles pour les *agents* interagissant avec le système. Chaque rôle permettant de brider les actions que peuvent entreprendre l'*agent*, limitant ou régissant ainsi ses impacts sur l'infrastructure à laquelle il appartient.

Le développement de l'API de **TuCSoN** est rythmée par les thèses et les travaux de recherches, dirigés notamment par **aliCE Research Group** [1]. Les dernières implications portant sur la modélisation de l'organisation et de la sécurité basée sur l'architecture RBAC (*Role-Based Access Control*) devraient être présentes dans la prochaine version.

Chapitre 5

Développement

Afin d'illustrer les fonctionnalités de l'API **TuCSoN** et d'appliquer les concepts fondamentaux issus de la compréhension du chapitre précédent, nous avons développé ici une application qui réagit à un flux d'information au sein d'une infrastructure.

L'idée de base est de déployer un système de règles dans un bâtiment contenant des capteurs, de manière à pouvoir gérer de façon efficace les cas où les mesures des grandeurs physiques donneraient lieu à des comportements non désirés. Chaque règle dont les termes sont confirmés par les mesures déclenchera une action qui sera gérée par la partie applicative du système et donnera une représentation visuelle à l'utilisateur.

Le système a été développé en **Java**. Trois exécutables (`.jar`) ont été réalisés. Le premier (`EnvironmentManager.jar`) permet de définir le bâtiment dans lequel le système est déployé, c'est-à-dire la définition des étages et des pièces dans lesquelles les capteurs sont installés. A chaque capteur est attribué un type de grandeur physique. Cet exécutable est aussi en charge de l'attribution des règles par un ensemble successif d'interfaces graphiques aidant l'utilisateur dans leur construction. Le second exécutable (`ActionManager.jar`) a pour but de gérer les différentes actions déclenchées par les règles. Il donne à l'utilisateur une indication précise (lieu, temps, valeur) de ce qui a satisfait la règle. Le troisième exécutable (`SensorEmulator.jar`) permet d'émuler les différents capteurs dans la bâtiment. Il permet, en effet, de simuler l'envoi d'une valeur simple ou d'une séquence aléatoire de valeurs.

La réalisation a été effectuée à l'aide de l'environnement de développement **NetBeans** et du **JDK1.6**. Pour les représentations graphiques plus complexes (tableaux - `JTable` et arbres - `JTree`), le pattern *Model-View-Controller* a été utilisé. Il est noté que les différents objets issus du package `javax.swing` englobent déjà les parties *View* et *Controller*, la partie *Model* a donc été implémentée manuellement, ce qui a permis de traduire un objet prédéfini (bâtiment ou règle, par exemple) vers l'élément graphique désiré.

5.1 Environnement de déploiement

L'environnement cible est un bâtiment dans lequel plusieurs capteurs sont installés. Chaque capteur donne un type de mesure unique à un intervalle régulier. Toutes les mesures sont évaluées par le système de manière à vérifier si une ou plusieurs règles sont satisfaites. Dans l'affirmative, une action qui aura été attribuée au préalable, sera déclenchée.

5.1.1 Bâtiment

Afin de calquer au mieux l'architecture sur l'infrastructure **TuCSoN** [1], le bâtiment dans lequel le système sera déployé va être hiérarchisé de la manière la plus logique possible.

En effet, en scindant le bâtiment en plusieurs étages, eux-mêmes scindés en plusieurs pièces, on parvient à dégager un système cartésien pour la "position" d'un capteur dans ce bâtiment. Un

1. Ce point sera détaillé dans la section suivante.

nom est attribué à chaque élément (bâtiment, étage, pièce). Le capteur, quant à lui, dispose d'un identifiant unique. La notion d'identifiant unique, dans une application concrète, est réaliste. En effet, en électronique par exemple, il est d'usage courant d'utiliser le numéro de série de la puce électronique comme identifiant (*Silicon Serial Number*). De manière syntaxique, on représente ces éléments par le listing 5.1.

$$
\begin{aligned}
&Building ::= BuildingName\ \&\ [Floor]_{1..i} \\
&Floor ::= FloorName\ \&\ [Room]_{1..j} \\
&Room ::= RoomName\ \&\ [Sensor]_{1..k} \\
&Sensor ::= UniqueId\ \&\ SensorType
\end{aligned}
$$

TABLE 5.1 – Représentation de l'environnement

Ces éléments sont traduits en **Java** (inclus dans l'annexe B) en quatre classes nommées successivement `Building.java`, `Floor.java`, `Room.java` et `Sensor.java`. Une cinquième classe énumère les différents types de capteurs utilisés dans l'application (`SensorType.java`). On a limité ici ces types à des grandeurs physiques communes : la température, l'humidité, la luminosité et la pression.

L'exécutable `EnvironmentManager.jar` permet de gérer le bâtiment et les éléments qui le constituent (dans l'onglet '`User`'). La figure 5.1 montre comment le bâtiment est représenté de façon hiérarchique avec un arbre (`JTree`). Lorsqu'un élément de l'arbre est sélectionné, l'ensemble des capteurs se trouvant dans cet élément est représenté dans le tableau (`JTable`) adjacent. Le capteur est ainsi présenté avec son type et son identifiant. Les fonctions d'ajout, d'édition et de retrait permettent de gérer et de construire l'objet bâtiment.

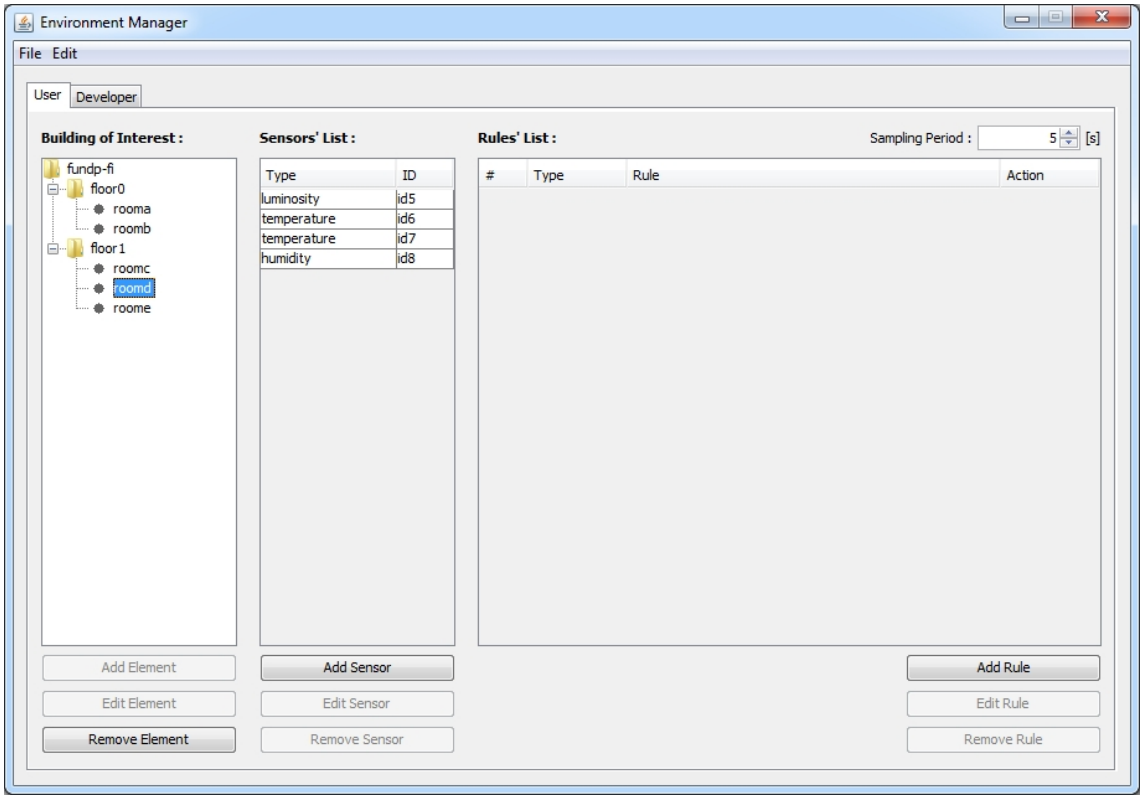

FIGURE 5.1 – EnvironmentManager - Premier onglet

5.1.2 Règles

Chaque valeur provenant d'un capteur peut être évaluée par une règle. L'évaluation de cette valeur peut se faire de plusieurs manières différentes : simple comparaison, inclusion dans un ensemble borné, règle composée, ou encore par une séquence. Cette section a pour but de montrer comment les règles sont construites et comment les informations sont gérées et évaluées. Il est à remarquer que les valeurs issues des capteurs sont envoyées à intervalle régulier appelé période d'échantillonnage. Une action est aussi attribuée à chaque règle. Ces actions seront définies dans la section suivante.

Règle simple

L'élément de base de la construction d'une règle simple est la comparaison de la valeur issue du capteur avec une constante. Cette constante peut être soit un nombre, soit une chaîne de caractères.
La comparaison avec un nombre utilise les opérateurs arithmétiques tels que : plus petit ($<$), plus petit ou égal ($\leq$), plus grand ($>$), plus grand ou égal ($\geq$), égal ($=$) et différent ($\neq$).
La comparaison avec une chaîne de caractères donne une représentation plus humanisée de la valeur du capteur. En effet, en définissant au préalable les termes utilisés, on peut établir comme règle que la température est élevée ou que la luminosité est faible. Ce genre de comparaison n'utilise que les opérateurs d'égalité ($=$ et $\neq$).

On définit un "corps de la règle" (`RuleBody`) comme étant le couple constitué d'une constante et d'un opérateur arithmétique. En créant une classe asbtraite `AbstractRuleBody<T>` (présentée en annexe C), on facilite l'implémentation de n'importe quel type de constante (nombre entier ou chaîne de caractères) utilisée pour la comparaison.
Pour l'application dont il est question ici, deux classes supplémentaires ont été implémentées : `IntegerRuleBody` et `StringRuleBody`. Elles définissent un corps de règle pour une comparaison avec un nombre entier et pour une chaîne de caractères en héritant de la classe asbtraite `AbstractRuleBody<T>`, où `T` est remplacée par `Integer` et `String` respectivement.

Une règle simple ne comportant qu'un seul corps de règle est appelée règle fixe (*fixed*). En définissant une règle simple composée de deux corps de règles, on construit une nouvelle règle pour laquelle la valeur issue d'un capteur n'est plus comparée à une seul constante, mais à deux constantes distinctes. Ce procédé met en place une règle vérifiant l'appartenance à un ensemble (*range*) de valeurs, borné par deux constantes, incluses ou non dans l'ensemble. Deux opérateurs mathématiques supplémentaires sont donc ajoutés : inclusion ($\in$) et exclusion ($\notin$).
La règle d'ensemble (*range*) est scindée en deux règles fixes. Suivant que l'on est face à une règle d'inclusion ou d'exclusion, l'implémentation est différente. En effet, dans le cas d'une inclusion, il faudra vérifier si la valeur issue du capteur est plus grande que (ou égale à) la borne inférieure ET plus petite que (ou égale à) la borne supérieure. Dans le cas d'une exclusion, par contre, il faudra vérifier si cette valeur est plus petite que (ou égale à) la borne inférieure OU plus grande que (ou égale à) la borne supérieure.

La classe `SimpleRule` présentée dans l'annexe C implémente les deux types de règles simples précitées. On ajoute comme élément de cette classe l'objet `ValueSource`, qui a pour but de définir d'où provient la valeur évaluée par la règle en question.

Règle composée

Une règle composée (`CompoundRule`) est une règle de plus haut niveau constituée d'une ou plusieurs règles simples appelées termes. Pour que cette règle soit satisfaite, il faut que tous les termes la constituant soient satisfaits et ce, endéans une seule période d'échantillonnage.
Il est à noter qu'une règle composée constituée d'un seul terme est assimilée à une règle simple.

Séquence

Une règle de type séquence est la règle de plus haut niveau (`Rule`), dans son sens le plus général. Une telle règle est constituée d'une ou plusieurs règles composées qui doivent être satisfaites successivement. On parle alors de règles composées satisfaisant les étapes de la règle de type séquence. En d'autres mots, à chaque étape, c'est-à-dire à chaque période d'échantillonnage, la règle composée relative à l'étape doit être satisfaite.

Une règle de type séquence constituée d'une seule étape est assimilée à une règle composée.

Le type de la règle que l'on construit est donc attribué en fonction du nombre d'étapes d'une séquence, du nombre de termes d'une règle composée et du nombre de corps de règles d'une règle simple. Le diagramme présenté à la figure 5.2 donne une représentation visuelle du raisonnement pour la définition du type de la règle ainsi construite.

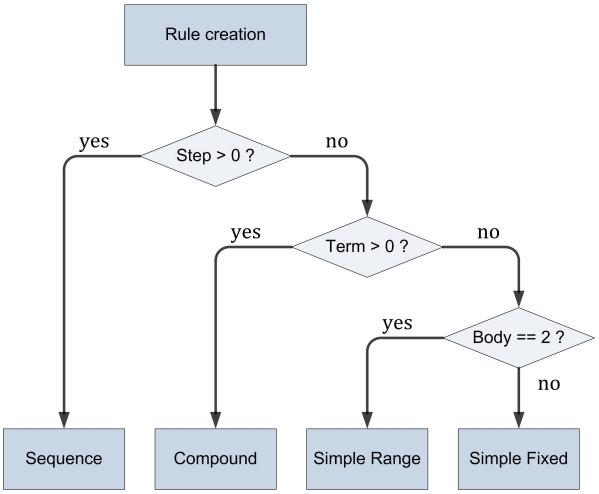

FIGURE 5.2 – Attribution du type de la règle

L'exécutable `EnvironmentManager.jar` permet de définir tous ces types de règles. Dans la figure 5.1, le bouton '**Add Rule**' en bas à droite de l'écran affiche la boîte de dialogue présentée à la figure 5.3. Cette boîte de dialogue permet à l'utilisateur de créer une règle parmi les possibilités affichées et d'y attribuer une action.

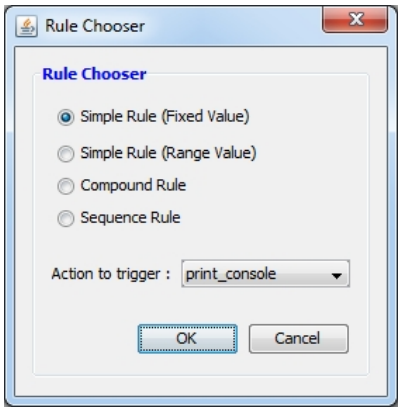

FIGURE 5.3 – Sélection du type de règle

Un exemple de règle simple est montré à la figure 5.4. Dans cet exemple, l'action associée à la règle sera déclenchée si la valeur de l'humidité donnée par le capteur présent dans la pièce `rooma` située à l'étage `floor0` est plus grande ou égale à 85% [2].

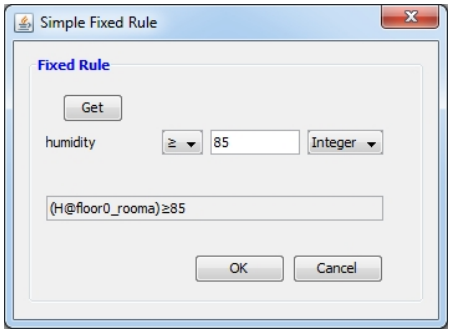

FIGURE 5.4 – Construction d'une règle simple (nombre entier)

Dans l'exemple de la figure 5.5, l'action associée à la règle sera déclenchée si la valeur de la température donnée par le capteur présent dans la pièce `rooma` située à l'étage `floor0` est égale à '`hot`'.

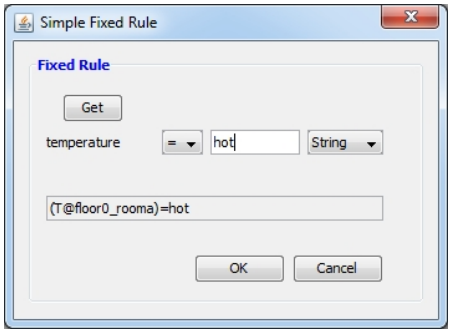

FIGURE 5.5 – Construction d'une règle simple (chaîne de caractères)

La règle présentée à la figure 5.6 est une règle d'ensemble (*range*). L'action associée à cette règle sera déclenchée si la valeur de la luminosité donnée par le capteur présent dans la pièce `roomc` située à l'étage `floor1` est comprise entre 200 lux et 2000 lux. La borne supérieure n'étant pas incluse dans l'ensemble.

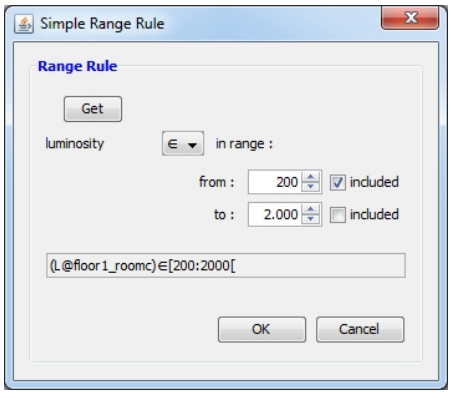

FIGURE 5.6 – Construction d'une règle simple (ensemble de valeurs)

2. Les unités ne sont pas présentées dans l'interface graphique.

Un exemple de règle composée est présenté à la figure 5.7. Dans cette boîte de dialogue, on peut ajouter des règles simples telles que décrites précédemment. Ici, la règle composée est constituée de deux termes qui doivent être satisfaits en même temps pour déclencher l'action associée à la règle.

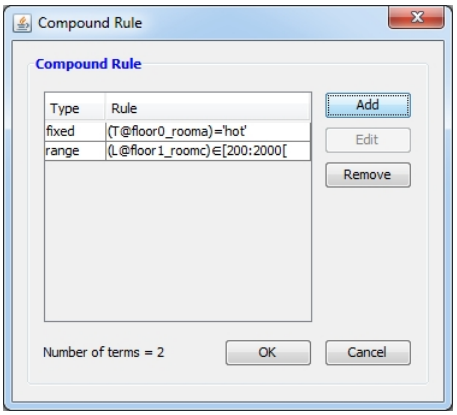

FIGURE 5.7 – Construction d'une règle composée

Le dernier exemple de construction d'une règle est présenté à la figure 5.8. Dans cette boîte de dialogue, on peut ajouter des règles simples ou des règles composées. La règle dont il est question ici est constituée de trois étapes, chacune devant être satisfaite successivement pour déclencher l'action associée à la règle.

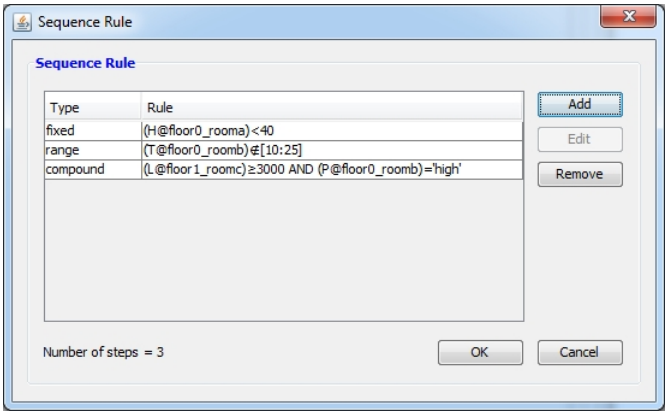

FIGURE 5.8 – Construction d'une règle de type séquence

Toutes les règles expliquées précédemment sont reprises dans la capture d'écran du programme `EnvironmentManager.jar` présenté à la figure 5.9. Pour plus de clarté, ces règles sont transcrites de manière syntaxique.

Un nombre entier est attribué à chaque règle. Ce nombre entier est un numéro unique issu de la classe `java.util.concurrent.atomic.AtomicInteger` qui s'incrémente à chaque fois qu'une nouvelle règle est créée. Si une règle est supprimée pendant l'exécution, son numéro sera perdu.

La période d'échantillonnage est paramétrée via le composant `JSpinner` en haut à droite de l'écran. Il est défini en secondes pour les démonstrations mais peut évidemment être fixé à 15 minutes ou 1 heure pour des applications plus réalistes.

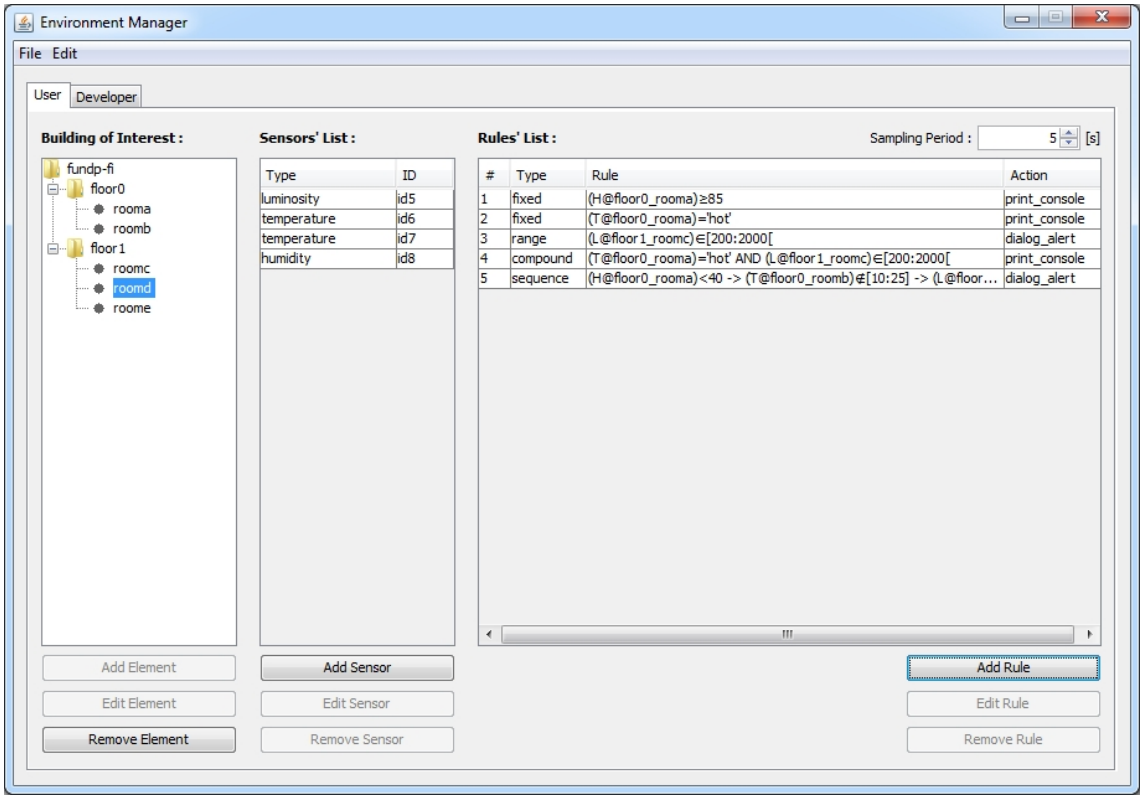

FIGURE 5.9 – EnvironmentManager - Présentation des règles

5.1.3 Actions

Les actions que l'on peut associer aux règles sont multiples et variées. Dans un système réel, on retrouve le déclenchement d'une alarme, l'envoi d'un e-mail ou d'un SMS, ou encore des actions prises sur la bâtiment en lui-même telles la fermeture d'une porte ou l'arrêt des ascenseurs. Dans tous les cas, c'est un ordinateur (ou un serveur) en charge de la gestion des actions qui commandera ces actions sur le monde réel grâce à des cartes d'entrées-sorties, d'une communication série (port COM, par exemple) ou d'un accès réseau.

Le système qui est implémenté ici ne gère pas d'action dans le monde réel. Au lieu de cela, il permet d'associer deux types d'actions prédéfinies : l'affichage des informations sur une console (`print_console`) ou dans une boîte de dialogue (`dialog_alert`). Les exemples de gestion et de déclenchement des règles seront présentés plus loin dans la section 5.6.

5.2 Architecture dans l'environnement TuCSoN

L'environnement, ou l'espace de coordination, de **TuCSoN** est constitué de plusieurs noeuds à l'intérieur desquels sont implémentés des *tuple centres*. Les informations transitent sous forme de *tuples* à travers ces *tuple centres* et ces noeuds.

Une des façons de calquer le bâtiment dans lequel sera implémenté le système de capteurs sur l'environnement **TuCSoN** est présentée dans le tableau 5.2.

Système	TuCSoN
Bâtiment	Noeud
Etage + Pièce	Tuple Centre
Type de capteur	Tuple

TABLE 5.2 – Bâtiment dans l'environnement TuCSoN

Un seul noeud sera installé par bâtiment. Le nom du *tuple centre* sera obtenu en concaténant le nom de l'étage et le nom de la pièce. Chaque *tuple* transportant l'information aura pour nom le type de capteur qui a envoyé l'information. Par exemple, pour un capteur de température (dont l'identifiant est `id1`) situé à l'étage `floor0` dans la pièce `rooma` envoyant la valeur 85, on aura dans l'espace de coordination **TuCSoN** le *tuple* suivant : `temperature(id1,85)` envoyé dans le *tuple centre* '`floor0_rooma`', situé dans le noeud installé localement dans le bâtiment. On obtiendra de la sorte un nom unique pour chaque *tuple centre* tel que requis par l'espace de coordination de **TuCSoN** (décrit à la section 4.3.2 à la page 19). Le fait de nommer le *tuple* par le type de capteur permet de regrouper plus facilement les règles suivant la grandeur physique mesurée. Un schéma global du système présentant les flux d'information est présenté à la figure 5.15 en fin de chapitre dans la section 5.7 en page 74.

L'exécutable `EnvironmentManager.jar` est scindé en deux onglets. Comme expliqué précédemment, le premier onglet ('`User`') a pour but de définir le bâtiment ainsi que les règles qui seront d'application. Le second onglet ('`Developer`') permet d'avoir une vue plus proche de l'implémentation **TuCSoN** (figure 5.10). L'arbre (`JTree`) à gauche de l'écran regroupe l'ensemble des *tuple centres* présents dans le noeud. Le tableau (`JTable`) adjacent liste toutes les réactions à programmer dans ces *tuple centres*, définies par les règles, par la gestion de la période d'échantillonnage (section 5.4) et par la fonctionnalité de monitoring (section 5.3). Le noeud **TuCSoN** doit être lancé avant toute action sur les *tuple centres*. Un *thread* en charge de l'installation du noeud (listing 5.1) est lancé à l'aide du bouton '`Launch Node`'. Lorsque le noeud est installé, s'il n'y a pas eu d'erreur, le statut "*TuCSoN Node launched ...*" est affiché. En cas d'erreur, une information est renvoyée à l'utilisateur.

```
package be.woine.thesis.utilz;

import alice.tucson.api.TucsonException;
import alice.tucson.service.Node;

/**
 * This class intends to create a thread handling a TuCSoN Node.
 * @author Remy
 * @version 1.0
 */
public class LaunchNode extends Thread {

    private Node node;

    /**
     * Constructor of the LaunchNode class.
     * @throws TucsonException Exception thrown by the TuCSoN Node.
     */
    public LaunchNode() throws TucsonException {
        node = new Node();
    }
```

```
    /**
     * Override of the run() method. This involves the node installation.
     */
    @Override
    public void run() {
        node.install();
    }

}
```

Listing 5.1 – LaunchNode Class

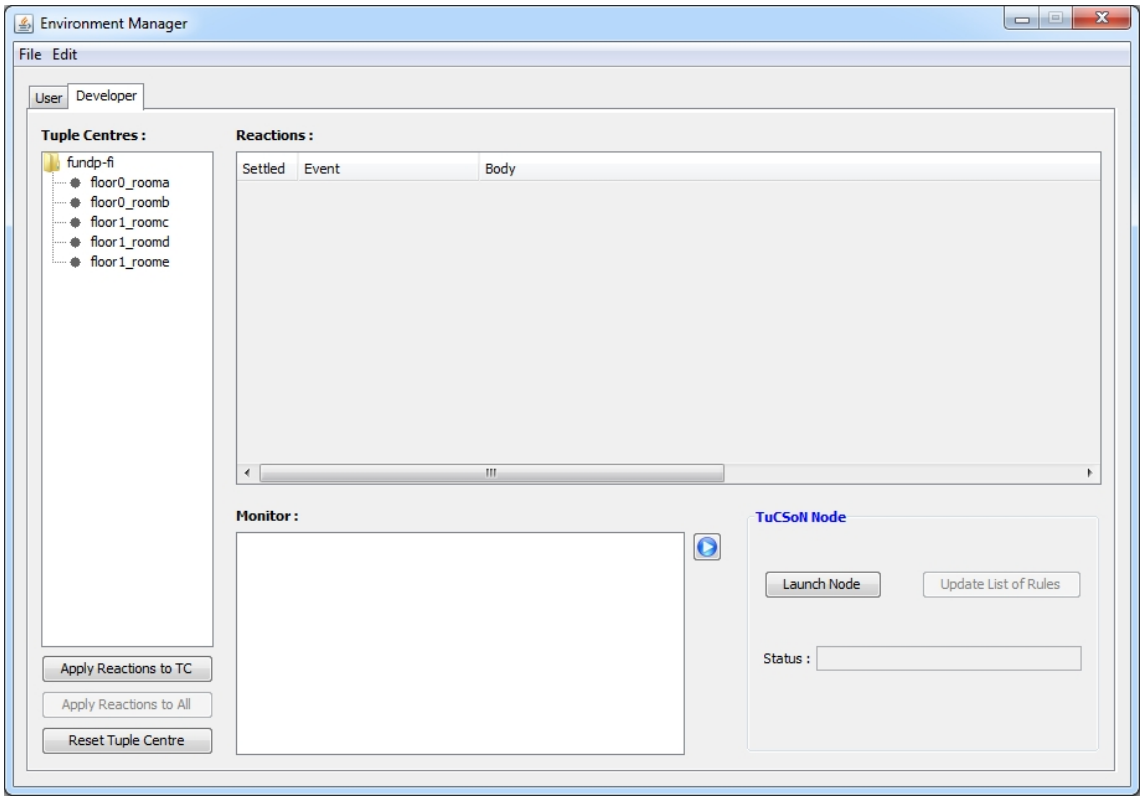

FIGURE 5.10 – EnvironmentManager - Deuxième onglet

5.3 Monitoring des informations

Un système de monitoring a été mis en place afin de surveiller tous les *tuples* entrant dans le système, c'est-à-dire tous les *tuples* envoyés par les capteurs. Les informations relatives au monitoring sont affichées dans la console présente en bas de l'écran du deuxième onglet ('`Developer`') de l'exécutable `EnvironmentManager.jar` (figure 5.10).

Afin d'implémenter un tel système, on spéficie dans chaque *tuple centre* du noeud, une réaction qui renverra le *tuple* vers un *tuple centre* spécifique.

```
reaction(out(X), (
    current_tc(N),
    current_tuple(T),
    out_tc(monitor_centre,monitor(N,T)) )).
```

Dans cette réaction, pour chaque *tuple* inséré (`out(X)`), on envoie au *tuple centre* spécifique `monitor_centre`, un *tuple* nommé `monitor` et constitué du *tuple* `T` qui a déclenché la réaction, ainsi que du nom `N` du *tuple centre* dans lequel le *tuple* `T` a été inséré.

La console de monitoring est alimentée par un *agent* qui surveille sans cesse l'arrivée de nouveaux *tuples* dans le *tuple centre* `monitor_centre`. L'implémentation de cet *agent* est montrée dans le listing 5.2, ou en version complète en annexe (listing E.1).

```
import alice.logictuple.*;
import alice.tucson.api.*;

public class MonitorAgent extends Thread {

  /*
  ...
  */

    public void run() {

    /* Create TuCSoN Context */
    TucsonContextSynch ctx = new TucsonContextSynch(Tucson.getContext(new AgentId(
        name)));

    /* Set Tuple Centre */
    TupleCentreId tcid = new TupleCentreId("monitor_centre");

    /* Construct Logic Template */
    LogicTuple template = new LogicTuple("monitor", new Var("TupleCentre"), new Var
        ("Tuple"));

    while(running) {

      /* Wait for tuple to come */
      LogicTuple req = ctx.in(tcid, template);

      /* Append Console  */
      // --> req.toString();

    }

    /* Exit Context */
    ctx.exit();
    }

}
```

Listing 5.2 – MonitorAgent Class

L'*agent* `MonitorAgent` est une classe qui hérite de la classe `Tread` de **Java**. Il est lancé par le programme `EnvironmentManager` lorsque l'utilisateur démarre le monitoring des *tuples* via le bouton '`play`', situé à droite de la console de monitoring. La figure 5.11 montre un exemple de l'arrivée de deux valeurs dans le *tuple centre* `floor0_rooma`. La date et l'heure qui sont affichées dans cette console proviennent d'une instance de la classe `java.util.Date` créée lors de l'arrivée de chaque *tuple*.

En sélectionnant l'élément `floor0_rooma` dans la liste de gauche, on peut observer l'ensemble des réactions relatives à ce *tuple centre*. La première réaction concerne le monitoring, tandis que les deux autres ont été implémentées pour la gestion de la période d'échantillonnage. Cette fonctionnalité est expliquée à la section suivante (5.4).

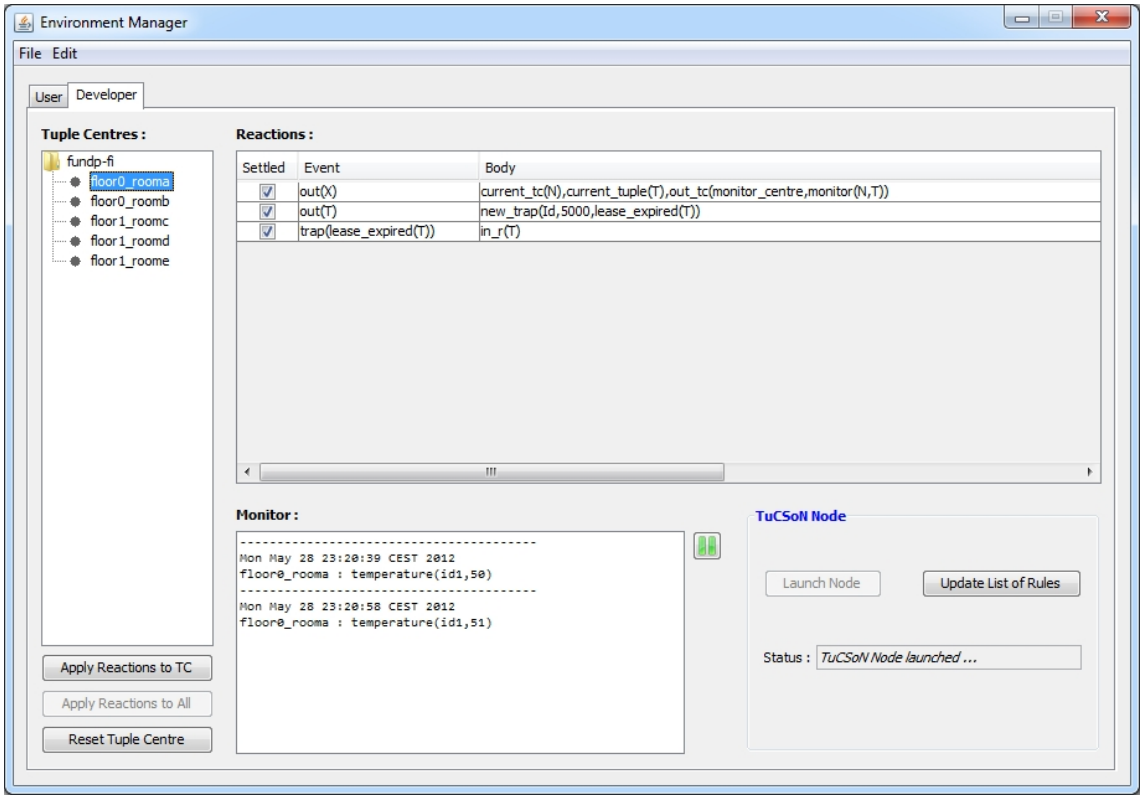

FIGURE 5.11 – EnvironmentManager - Console de monitoring (exemple)

5.4 Gestion de la période d'échantillonnage

Vu que les *tuples* arrivent à intervalle régulier (suivant la période d'échantillonnage) dans le système, il faut continuellement purger les *tuple centres* des anciens *tuples*. Pour ce faire, on implémente une version allégée de la notion de leasing telle que décrite dans la sous-section 4.6.3.1 à la page 49. En effet, ici, chaque *tuple* qui arrive est évalué de suite par les réactions relatives aux règles instaurées par l'utilisateur. Il n'y a donc pas lieu d'implémenter les reactions de leasing relatives aux primitives de communication `in` et `rd` sur le *tuple* en question. La réaction se simplifie donc à :

```
reaction(out(T), (
    new_trap(Id,SamplingTime,lease_expired(T)) )).

reaction(trap(lease_expired(T)), (
    in_r(T) )).
```

où l'on crée une nouvelle `trap` dès l'arrivée d'un *tuple* (première réaction). La variable indiquant le temps (`SamplingTime`) doit être remplacée par la période d'échantillonnage en millisecondes. La seconde réaction enlève le *tuple* grâce à la primitive de communication `in_r` lorsque le temps préalablement spécifié est expiré. On peut observer ces deux réactions pour le *tuple centre* `floor0_rooma` à la figure 5.11.

5.5 Mise en place des règles

Les règles construites à l'aide de l'exécutable `EnvironmentManager.jar` sont traduites par un ensemble de réactions qui seront programmées dans les *tuple centres* de l'espace de coordination de **TuCSoN**. Suivant le type de règle auxquelles on a affaire, l'implémentation est différente. Les sections suivantes reprennent le développement de la traduction de chaque type de règle en un ensemble de réactions écrites dans le langage **ReSpecT**.

Certains aspects sont cependant communs à toutes les règles. En effet, lorsqu'une règle est satisfaite, elle déclenche une action. Cette action doit non seulement agir sur l'environnement, mais aussi informer un utilisateur de l'ensemble des causes qui a satisfait la règle. A noter qu'en cas de règles composées ou des règles de type séquence, c'est un ensemble de règles qui doit être satisfait pour déclencher l'action. En résumé, pour chaque règle satisfaite, on renverra à l'utilisateur (ici, l'exécutable `ActionManager.jar`) le numéro unique de la règle, l'identifiant du capteur qui a satisfait la règle et la valeur mesurée par celui-ci.

Afin de faciliter la gestion des informations, on fixera une syntaxe (listing 5.3) pour le *tuple* contenant les valeurs envoyées par les capteurs. Ce *tuple* reprendra le type du capteur, son identifiant et la valeur mesurée.

out(sensor_type(`Id`,`Value`))

TABLE 5.3 – Syntaxe du tuple du capteur

Pour les actions déclenchées par les règles, on conviendra du *tuple centre* `action_centre` dans lequel des *tuples* spécifiques seront envoyés. Ces *tuples* avertiront l'agent en charge des actions qu'une action doit être gérée. Le listing 5.4 reprend la syntaxe générique du prédicat.

out_tc(action_centre,action(`RuleId`,ids([`IdL`]),values([`ValueL`])))

TABLE 5.4 – Syntaxe du prédicat de l'action

La primitive **ReSpecT** `out_tc` permet l'envoi d'un *tuple* vers le *tuple centre* mentionné. On envoie donc dans le *tuple centre* `action_centre`, le *tuple* `action` composé du numéro unique de la règle qui a déclenché l'action ainsi que de deux *tuples* supplémentaires. Le premier (`ids`) contenant la liste des identifiants des capteurs, le second (`values`) contenant la liste des valeurs données par les capteurs. Ces deux *tuples* ont pour but de renvoyer des informations complètes à l'*agent* en charge de la gestion des actions. La construction de ces listes est expliquée dans les sous-sections suivantes.

5.5.1 Règle simple (fixe)

Une règle simple avec une valeur fixe est la plus facile à traduire, en ce sens qu'elle ne comporte que très peu d'information à gérer. Cette règle servira toutefois de base pour la construction des règles plus complexes.

Dans la réaction **ReSpecT**, l'***Event*** sera écrit en fonction du type de capteur, c'est-à-dire du type de grandeur physique sur laquelle la règle a été établie. Le ***Body*** de la réaction, quand à lui, comportera la comparaison mathématique et la gestion de l'action associée à la règle. Le listing 5.5 présente de manière syntaxique ce qu'on obtient pour la traduction d'une règle simple avec une constante.

SimpleRuleReaction : := reaction(*Event*, (*Body*)). *Event* : := out(sensor_type(`Id`,`Value`)) *Body* : := *SimpleRule*, *ActionMgt* *SimpleRule* : := `Value Op Constante` *ActionMgt* : := out_tc(action_centre,action(`RuleId`,ids([`Id`]),values([`Value`])))

TABLE 5.5 – Syntaxe d'une règle simple (fixe)

Afin d'illustrer cette syntaxe, on prend l'exemple suivant : une règle dont le numéro unique est 1 déclenche une action lorsque la valeur de la température est supérieure à 50 degrés.

```
reaction(out(temperature(Id,T)), (
    T>50,
    out_tc(action_centre,action(1,ids([Id]),values([T]))) )).
```

La règle simple ne portant que sur un seul capteur, les listes contenues dans le *tuple* `action` ne comportent, chacune, qu'un seul élément. On garde toutefois la notion de liste afin de dégager une certaine généricité dans la gestion des informations du côté de l'*agent* en charge des actions.

5.5.2 Règle simple (ensemble)

Une règle simple portant sur un ensemble de valeurs est construite sur base de deux règles simples avec une valeur fixes déclenchant une même action. De plus, suivant que la règle d'ensemble porte sur une inclusion ou une exclusion, on devra écrire respectivement une ou deux réactions. Les listings 5.6 et 5.7 présentent une vue syntaxique de ces deux manières de coder.

SimpleRuleReaction : := reaction(*Event*, (*Body*)). *Event* : := out(sensor_type(`Id`,`Value`)) *Body* : := *SimpleRule*, *ActionMgt* *SimpleRule* : := `Value Op1 LowerBound`, `Value Op2 UpperBound` *ActionMgt* : := out_tc(action_centre,action(`RuleId`,ids([`Id`]),values([`Value`])))

TABLE 5.6 – Syntaxe d'une règle simple (ensemble - inclusion)

SimpleRuleReaction : := reaction(*Event*, (*Body*)). *Event* : := out(sensor_type(`Id`,`Value`)) *Body* : := *SimpleRule*, *ActionMgt* *SimpleRule* : := `Value Op1 LowerBound` *ActionMgt* : := out_tc(action_centre,action(`RuleId`,ids([`Id`]),values([`Value`])))

SimpleRuleReaction : := reaction(*Event*, (*Body*)). *Event* : := out(sensor_type(`Id`,`Value`)) *Body* : := *SimpleRule*, *ActionMgt* *SimpleRule* : := `Value Op2 UpperBound` *ActionMgt* : := out_tc(action_centre,action(`RuleId`,ids([`Id`]),values([`Value`])))

TABLE 5.7 – Syntaxes d'une règle simple (ensemble - exclusion)

Pour la règle d'inclusion, le fait de mettre dans une seule réaction les deux comparaisons mathématiques force la réaction de tenir compte du résultat de ces deux comparaisons avant de déclencher l'action (ET logique). Pour la règle d'exclusion, on a traduit en deux réactions distinctes de manière à effectuer un OU logique sur les deux comparaisons mathématiques. Les exemples qui suivent illustrent les deux types de règles simples d'ensemble.

- Supposons qu'une règle dont le numéro unique est 2 déclenche une action lorsque la valeur de l'humidité est comprise entre 10% (inclus) et 45% (exclus). La traduction **ReSpecT** de la réaction est donnée par :

```
reaction(out(humidity(Id,H)), (
    H>=10,
    H<45,
    out_tc(action_centre,action(2,ids([Id]),values([H]))) )).
```

- Supposons qu'une règle dont le numéro unique est 3 déclenche une action lorsque la valeur de l'humidité n'est pas comprise entre 45% (exclus) et 85% (inclus). La traduction **ReSpecT** des réactions est donnée par :

```
reaction(out(humidity(Id,H)), (
    H=<45,
    out_tc(action_centre,action(3,ids([Id]),values([H]))) )).

reaction(out(humidity(Id,H)), (
    H>85,
    out_tc(action_centre,action(3,ids([Id]),values([H]))) )).
```

Ces règles ne portant aussi que sur un seul capteur, les listes contenues dans le *tuple* `action` ne comportent donc, chacune, qu'un seul élément.

5.5.3 Règle composée

Une règle composée peut contenir plusieurs règles simples, chacune pouvant porter sur une valeur fixe ou sur un ensemble. Une règle composée est satisfaite si toutes les règles (appelés termes) qui la composent sont satisfaites endéans une seule période d'échantillonnage. Il faudra donc gérer chaque terme et inclure une notion du temps dans les réactions.

Le déclenchement de l'action n'est pas aussi directe que pour les règles simples. Ici, chaque règle simple qui constitue la règle composée va être codé de la façon décrite dans les sous-sections 5.5.1 et 5.5.2, à la différence qu'on n'enverra plus de *tuples* `action` vers le *tuple centre* `action_centre`, mais plutôt des *tuples* `term` dans un autre *tuple centre* spécifique à la règle composée appelé `compound_#`. La dièse (`#`) prendra comme valeur le numéro unique de la règle composée (attribué via la classe `java.util.concurrent.atomic.AtomicInteger` lors de la création de la règle). La syntaxe du prédicat est présenté au listing 5.8. La variable `X` du *tuple* `term` permet d'identifier chaque sous-règle de la règle composée.

out_tc(compound_#,term(X,Id,Value))

TABLE 5.8 – Syntaxe du prédicat pour la règle composée

Afin de mieux se représenter le déroulement des opérations, on va considérer l'exemple suivant : une action est déclenchée lorsque la température dépasse 50 degrés et que la luminosité dépasse 20000 lux. La règle composée qui regroupe ces sous-règles porte le numéro unique 4. Les deux réactions codées pour les *tuple centres* relatifs aux mesures de température et luminosité sont représentés ci-après.

```
reaction(out(temperature(Id,T)), (
    T>50,
    out_tc(compound_4,term(0,Id,T)) )).

reaction(out(luminosity(Id,L)), (
    L>20000,
    out_tc(compound_4,term(1,Id,L)) )).
```

Dans cette exemple, le *tuple centre* `compound_4` sera donc alimenté par des *tuples* de la forme `term(X,Id,Value)`. Le nombre de termes correspondant donc au nombre de sous-règles de la règle composée. A chaque fois que ce type de *tuple* est inséré, on va écrire les informations qu'il transporte dans trois autres *tuples* dédiés respectivement aux termes (`all_term`), aux identifiants (`all_id`) et aux valeurs (`all_value`). Il est important de noter que l'ordre d'arrivée de ces *tuples* n'est pas déterministe.

Les tuples `all_term`, `all_id` et `all_value` contiennent chacun un nombre de variables égal au nombre de termes. L'indice de la variable correspondant au numéro du terme. Le but étant, pour satisfaire la règle, que le *tuple* `all_term` contienne le numéro de chaque terme : `all_term(0,1)`. Il en va de même pour les *tuples* `all_id` et `all_value`. On codera donc une réaction qui va alimenter ces *tuples* chaque fois qu'un *tuple* `term` est inséré dans le *tuple centre* `compound_#`. En reprenant l'exemple précédent, on peut écrire les deux réactions suivantes :

```
reaction(out(term(0,Id,Value)), (
    in_r(term(0,Id,Value)),
    in_r(all_term(X0,X1)),
    out_r(all_term(0,X1)),
    in_r(all_id(I0,I1)),
    out_r(all_id(Id,I1)),
    in_r(all_value(V0,V1)),
    out_r(all_value(Value,V1))
     )).

reaction(out(term(1,Id,Value)), (
    in_r(term(1,Id,Value)),
    in_r(all_term(X0,X1)),
    out_r(all_term(X0,1)),
    in_r(all_id(I0,I1)),
    out_r(all_id(I0,Id)),
    in_r(all_value(V0,V1)),
    out_r(all_value(V0,Value))
     )).
```

En reprenant chaque prédicat dans l'ordre, on a dans le ***Body*** de la réaction :

- Enlèvement du *tuple* qui a déclenché la réaction.
- Enlèvement du *tuple* `all_term`.
- Ecrasement de l'indice correspondant au numéro du terme.
- Enlèvement du *tuple* `all_id`.
- Ecrasement de l'identifiant avec l'identifiant (`Id`) du *tuple* qui a déclenché la réaction.
- Enlèvement du *tuple* `all_value`.
- Ecrasement de la valeur avec la valeur (`Value`) du *tuple* qui a déclenché la réaction.

Afin de ne pas faire échouer la réaction avec les prédicats bloquants `in_r`, on doit s'assurer qu'il existe bien des *tuples* `all_term`, `all_id` et `all_value` dans le *tuple centre* `compound_4`. On va donc ajouter une réaction qui insérera ces *tuples* initialisés dès qu'un des termes est inséré dans le *tuple centre*. Cette réaction est reprise ci-après.

```
reaction(out(term(_,_,_)), (
    no_r(all_term(_,_)),
    out_r(all_term(-1,-1)),
    out_r(all_id(-1,-1)),
    out_r(all_value(-1,-1))
     )).
```

Avant d'aller plus loin, on doit s'assurer que tous les termes arrivent endéans une seule période d'échantillonnage. Si ce n'est pas le cas, on devra réinitialiser le *tuple centre* de manière à ce qu'il ne contienne que les *tuples* `all_term`, `all_id` et `all_value` initialisés comme dans la réaction précédente. On va donc utiliser la notion de `trap` telle que présentée dans la section 4.6.3.

```
reaction(out(term(X,Y,Z)), (
    no_r(first),
    out_r(first),
    new_trap(Id,SamplingTime,lease_expired(term(X,Y,Z))),
    out_r(trap_id(Id))
     )).

reaction(trap(lease_expired(term(X,Y,Z))), (
    in_r(all_term(_,_)),
    in_r(all_id(_,_)),
    in_r(all_value(_,_)),
    out_r(all_term(-1,-1)),
    out_r(all_id(-1,-1)),
    out_r(all_value(-1,-1)),
    out_r(trap_id(Id)),
    in_r(first),
    in_r(trap_id(Id))
     )).
```

Dans la première réaction on s'assure qu'il n'y ait qu'une `trap` installée dans le *tuple centre*, c'est la raison pour laquelle on utilise un *tuple* `first` comme '*token*'. S'il est déjà présent dans le *tuple centre*, la réaction échoue. Dans le cas contraire, on insère le *tuple* `first` et on installe une nouvelle `trap`. L'identifiant de cette `trap` est "publié" dans le *tuple centre* grâce au *tuple* `trap_id`. L'utilité sera expliquée un peu plus loin dans le document.

La deuxième réaction est déclenchée lorsque le temps ***SamplingTime*** mentionné lors de la création de la `trap`, égal à la période d'échantillonnage, est écoulé. La réaction déclenche donc le retrait de tous les *tuples* présents et réinitialise le *tuple centre* avec les *tuples* par défaut.

La dernière réaction à programmer dans le *tuple centre* `compound_#` concerne le déclenchement de l'action associée à la règle. Comme expliqué précédemment, cette action doit être déclenchée lorsque le *tuple* `all_term` contient tous les indices des termes. Un nouveau *tuple* `action` doit alors être envoyé au *tuple centre* `action_centre`. Ce *tuple* doit contenir le numéro de la règle ainsi que les listes contenant respecivement l'identifiant de chaque capteur et la valeur qu'il a mesurée. Cette réaction est montrée ci-dessous.

```
reaction(out(all_term(0,1)), (
    in_r(all_term(0,1)),
    in_r(all_id(I0,I1)),
    in_r(all_value(V0,V1)),
    IdList=[I0,I1],
    ValueList=[V0,V1],
```

```
out_r(all_term(-1,-1)),
out_r(all_id(-1,-1)),
out_r(all_value(-1,-1)),
rd_r(trap_id(Id)),
kill_trap(Id),
in_r(first),
in_r(trap_id(Id)),
out_tc(action_centre, action(4,ids(IdList),values(ValueList)))
 )).
```

Dans cette réaction, on commence par lire/enlever tous les *tuples* contenant les informations des termes : `all_term`, `all_id` et `all_value`. Les listes des identifiantes (***IdList***) et des valeurs (***ValueList***) sont ensuite créées. Les trois prédicats suivant permettent de réinitialiser les *tuples* à leurs états par défaut. Ensuite, on va lire l'identifiant de la `trap` qui a été créée auparavant afin de l'éliminer via le prédicat `kill_trap`. Les deux *tuples* `first` et `trap_id` sont ensuite enlevés du *tuple centre*. Enfin, le dernier prédicat envoie dans le *tuple centre* `action_centre`, le *tuple* `action` contenant toutes les informations nécessaires à la gestion de l'action associée à la règle.

5.5.4 Séquence

Les règles de type séquence n'ont pas été implémentées dans l'application. Toutefois, on peut décrire quelles auraient été les réactions si on avait dû implémenter de telles réactions.

La gestion des séquences, tout comme la gestion des règles composées, doit se gérer en plusieurs étapes. Par contre, ici, une notion de temps doit apparaître pour chaque étape. Et non plus globalement, comme c'est le cas pour l'ensemble des termes des règles composées.

On peut convenir d'un *tuple centre* spécifique qui sera alimentée par chaque étape de la séquence. L'insertion des *tuples* se fera au moyen du prédicat présenté dans le listing 5.9.

out_tc(sequence_#,step(X,IdList,ValueList))

TABLE 5.9 – Syntaxe du prédicat pour la séquence

Le *tuple centre* prendra comme nom `sequence_#`, où la dièse sera remplacée par le numéro unique de la règle. Il recevra des *tuples* sous la forme `step`, reprenant le numéro de l'étape de la séquence (`X`), la liste des identifiants (`IdList`) des capteurs qui ont déclenché la règle et la liste des valeurs mesurées (`ValueList`), en rapport avec la liste précédente. Dans le cas où les étapes sont composées par des règles simples, les listes ne contiendront qu'un seul élément.

Chaque *tuple* `step` arrivant dans le *tuple centre* `sequence_#` ne devra y rester que pour une seule période d'échantillonnage (installation d'une `trap` lors de l'arrivée du *tuple*). Lorsqu'une autre étape arrive, on vérifiera si l'étape précédente est toujours présente dans le *tuple centre*. Dans l'affirmative, le *tuple* précédent est enlevé et une nouvelle `trap` sera installée pour le *tuple* actuel. Dans le cas contraire, tous les *tuples* sont enlevés et le *tuple centre* est réinitialisé à son état par défaut.

Si les étapes arrivent bien à chaque période, on parviendra à déclencher le *tuple* relatif à l'action lors de la dernière étape. Les informations concernant les identifiants des capteurs et de leur mesure respective devront être stockées dans des listes. Ces listes seront construites à chaque étape de la règle.

5.6 Gestion des actions

Les actions déclenchées par la satisfaction des règles sont gérées par un autre exécutable appelé `ActionManager.jar`. La figure 5.12 montre une capture d'écran lorsque l'*agent* est démarré. On retrouve dans cette interface, une case à cocher (`JCheckBox`) permettant de définir si l'*agent* en charge des actions est localisé sur la même machine (`'localhost'`) que l'exécutable `EnvironmentManager.jar` ou sur un ordinateur distant. Dans un tel cas, l'adresse IP de l'ordinateur qui a démarré le noeud **TuCSoN**, c'est-à-dire celui qui a exécuté `EnvironmentManager.jar`, doit être mentionnée. La fenêtre de *log* donne une indication en temps réel sur le travail qu'exécute l'*agent*.

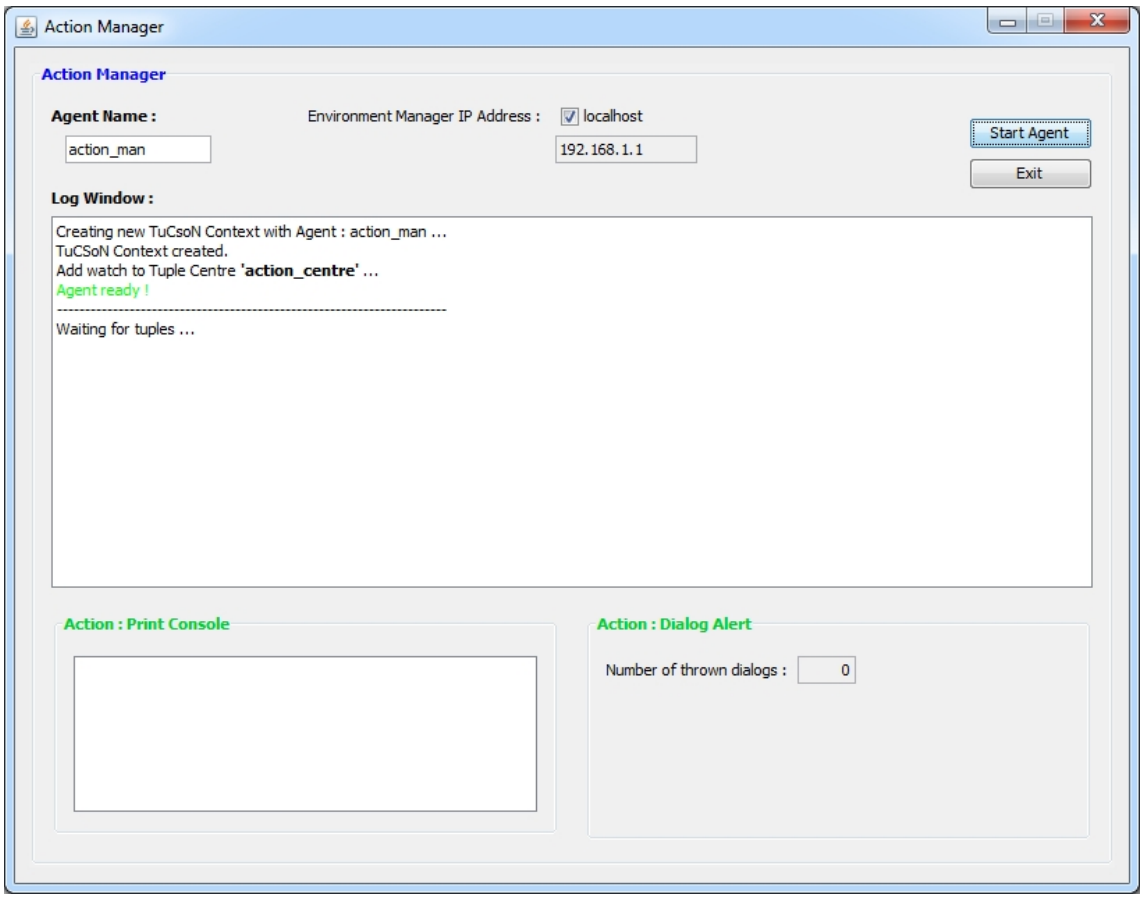

FIGURE 5.12 – ActionManager - Démarrage de l'agent

Dans la partie inférieure de cette fenêtre se trouve des conteneurs visuels relatifs aux actions déclenchées par les règles. Pour rappel, dans le cadre de ce travail, ces actions sont de deux types : `print_console` et `dialog_alert`. Si l'action déclenchée demande un affichage console, un descriptif de la règle sera présenté dans la zone de texte (`JTextArea`) en bas à gauche de la fenêtre. Par contre, dans le deuxième cas, une boîte de dialogue sera affichée à l'écran. Cette petite fenêtre affichera une description plus embellie de la règle. Le nombre de boîte de dialogues est affiché en bas à droite de la fenêtre principale de l'interface `ActionManager.jar`.

A l'image de l'*agent* en charge du monitoring, l'*agent* en charge des actions va sans cesse scruter l'arrivée de nouveaux *tuples* dans le *tuple centre* qui lui est attribué : `action_centre`. L'implémentation de cet *agent* est montrée dans le listing 5.3, ou en version complète en annexe (listing E.2).

```
import alice.logictuple.*;
import alice.tucson.api.*;

public class ActionAgent extends Thread {

  /*
  ...
  */

    public void run() {

        /* Create TuCSoN Context */
    TucsonContextSynch ctx = new TucsonContextSynch(Tucson.getContext(new AgentId(
        name)));

    /* Set Tuple Centres */
    TupleCentreId tcid = new TupleCentreId("action_centre@" + nodeAddress);
    TupleCentreId tcid_ex = new TupleCentreId("exchange_centre@" + nodeAddress);

    /* Construct Logic Templates */
    LogicTuple template = new LogicTuple("action", new Var("RuleId"), new Var("
        IdList"), new Var("ValueList"));
    LogicTuple template_ex = new LogicTuple("rule", new Var("RuleId"), new Var("
        Object"));

    while(running) {

      /* Wait for action tuple to come */
      LogicTuple req = ctx.in(tcid, template);

      /* Send request with Rule ID */
      ctx.out(tcid_ex, new LogicTuple("rule_req", req.getArg(0)));

      /* Wait for response */
      LogicTuple rule_req = ctx.in(tcid_ex, template_ex);

      /* Deserialize the received tuple (rule_req.getArg(1)) to get a Rule object
          */
      /* Use 'ByteArrayInputStream' and 'ObjectInputStream' */

      /* Get Lists from first received tuple */
      LogicTuple idsTuple = LogicTuple.parse(req.getArg(1).toString());
            List idList = idsTuple.getArg(0).toList();

      LogicTuple valuesTuple = LogicTuple.parse(req.getArg(2).toString());
            List valueList = valuesTuple.getArg(0).toList();

      /* Process Action */
      processAction(rule, idList, valueList);
    }
  }

  ctx.exit();

}
```

Listing 5.3 – ActionAgent Class

Lorsque l'agent a reçu dans le *tuple centre* `action_centre` un *tuple* `action`, le seul moyen qu'il a d'obtenir de manière complète la règle qui a déclenché la réaction, est de demander l'objet `Rule` créé par `EnvironmentManager.jar`. L'*agent* en charge de l'action va donc envoyer une requête spécifique dans le *tuple centre* `exchange_centre` afin que l'exécutable `EnvironmentManager.jar` lui envoie une version sérialisée de l'objet `Rule` dont l'identifiant mentionné à déclenché l'action. Du côte de `EnvironmentManager.jar`, un *agent* (un *thread*) scrute en permanence l'arrivée d'une requête dans ce *tuple centre*.

La sérialisation est réalisée grâce aux classes du package `java.io` : `ByteArrayOutputStream` et `ObjectOutputStream`. L'objet ainsi sérialisé en transformé en un tableau de bytes qui sera encapsulé dans un tuple avant d'être envoyé à l'autre agent. Du côté de la désérialisation, on utilise les classes correspondantes, à savoir : `ByteArrayInputStream` et `ObjectInputStream`. Il est à noter que les classes contenues dans les packages `be.woine.thesis.rule.common` (annexe C), `be.woine.thesis.building.common` (annexe B) et `be.woine.thesis.action.common` (annexe D) implémentent toutes l'interface `java.io.Serializable`.

La capture d'écran présenté à la figure 5.13 montre le délenchement de deux actions (une de chaque type). On remarque le déroulement du processus entrepris par l'*agent* dans sa fenêtre de log. Lorsqu'un premier *tuple* arrive, il est affiché tel quel dans la fenêtre de log. Ensuite, l'*agent* va exécuter une requête ("*Processing ...*") pour obtenir la règle correspondant à l'identifiant reçu. Cette règle est affichée de manière synthétique dans la fenêtre de log, mais de manière détaillée dans la console en bas de la fenêtre. La deuxième règle déclenche quant à elle, une boîte de dialogue (figure 5.14).

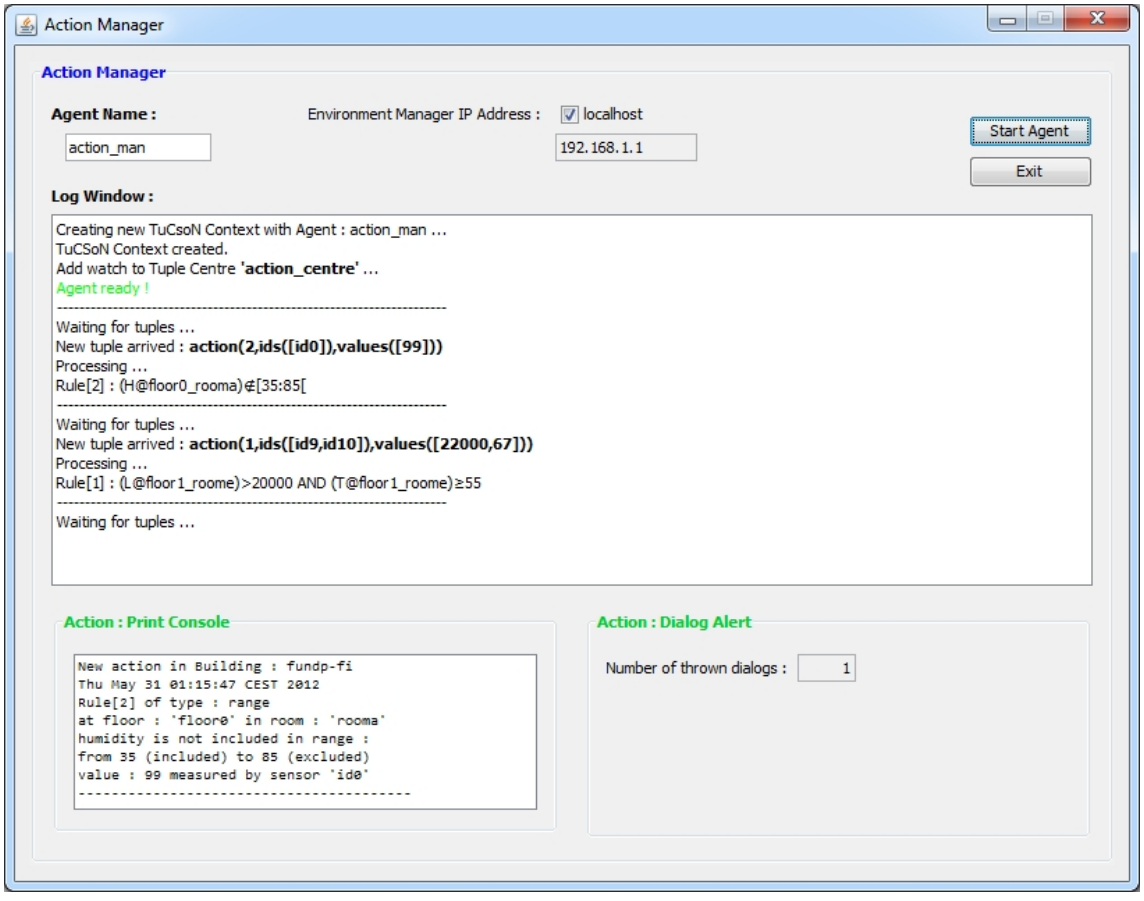

FIGURE 5.13 – ActionManager - Traitement des actions

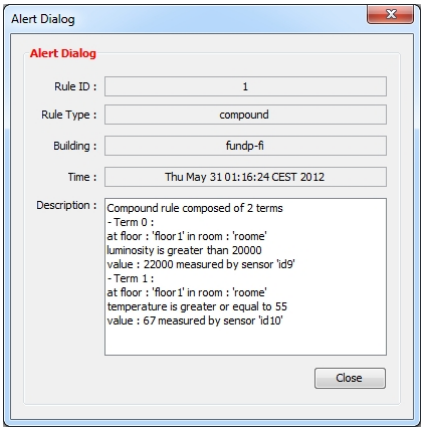

FIGURE 5.14 – ActionManager - Boîte de dialogue

5.7 Alimentation du système

Le diagramme présenté à la figure 5.15 montre une vue complète de l'échange d'information dans le système.

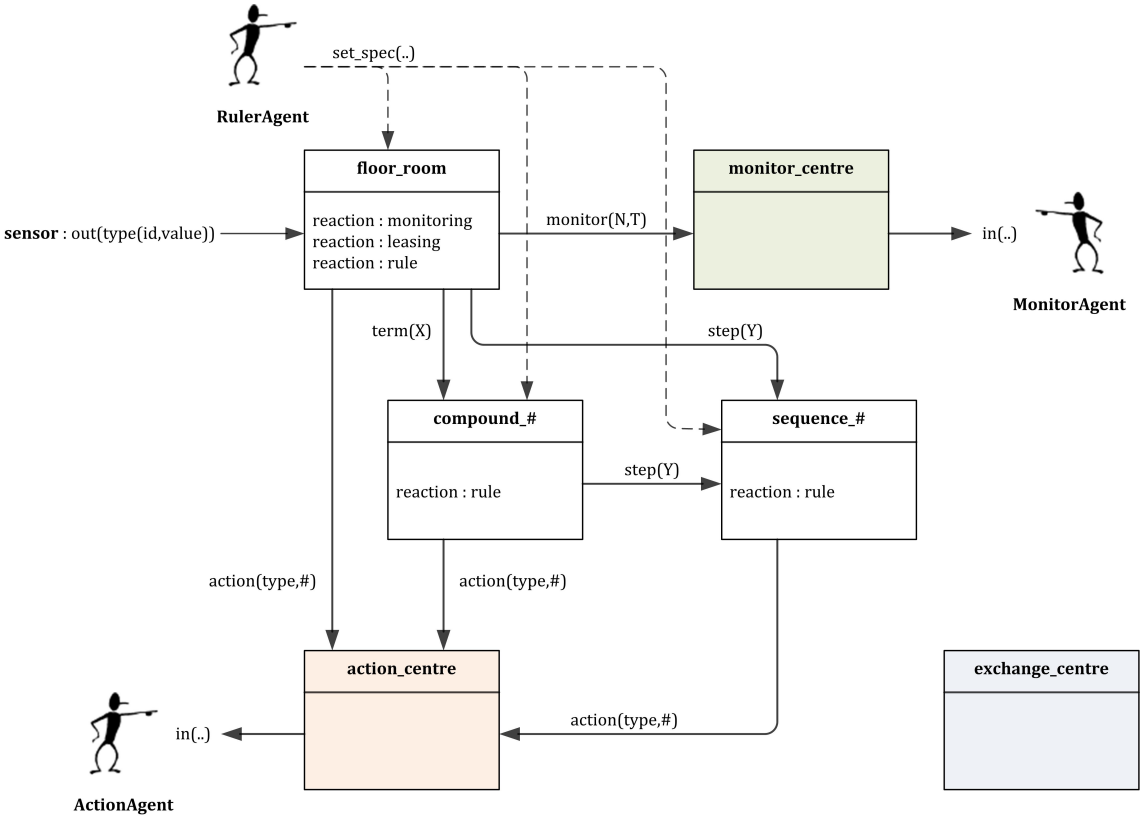

FIGURE 5.15 – Flux d'informations dans le système

Chaque capteur envoie un *tuple*, nommé par le type de grandeur physique et constitué de son identifiant et de la valeur mesurée, au *tuple centre* correspondant à l'endroit où il se trouve (couple étage-pièce). Ce *tuple* est directement lu et envoyé au *tuple centre* `monitor_centre` en charge du monitoring (réaction du monitoring). La gestion du *leasing* avec la période d'échantillonnage est gérée directement dans le *tuple centre* d'arrivée du *tuple* (réaction du leasing). Enfin, les réactions relatives aux règles instaurées permettent d'aiguiller les *tuples* de manière adéquate :

– Si le *tuple* satisfait une règle simple, une règle composée complète ou une règle de type séquence complète, il déclenche une action grâce à l'envoi d'un *tuple* spécifique dans le *tuple centre* `action_centre`.
– Si le *tuple* satisfait un terme d'une règle composée, il déclenche l'envoi d'un *tuple* spécifique nommé `term` au *tuple centre* correspondant à la règle composée (`compound_#`).
– Si le *tuple* satisfait une étape d'une règle de type séquence, il déclenche l'envoi d'un *tuple* spécifique nommé `step` au *tuple centre* correspondant à la règle de type séquence (`sequence_#`).

Les actions sont envoyées via des *tuples* spécifiques au *tuple centre* `action_centre`. Ces actions sont interprétées et gérées par l'*agent* en charge de ce *tuple centre*.

Un *tuple centre* dédié nommé `exchange_centre` permet l'échange d'information entre les *agents*. Il s'agit ici de faire passer des objets **Java** sérialisés entre ces *agents*.

L'idée première lors de l'attribution du sujet de ce mémoire, était d'alimenter le système avec un réseau de capteurs sans fil déployé dans un bâtiment. Malheureusement, le temps à jouer contre nous et le matériel n'a pas su être mis à disposition. Il a donc fallu trouver un autre moyen pour envoyer des informations à traiter au système. C'est la raison pour laquelle l'exécutable `SensorEmulator.jar` (présenté dans la sous-section 5.7.1) a été développé. Tous les exemples illustrés dans ce chapitre utilise une émulation des capteurs.

5.7.1 Emulation

L'exécutable `SensorEmulator.jar` permet de simuler l'envoi d'information de n'importe quel capteur qui a été préalablement défini par l'`EnvironmentManager.jar`. A la manière de l'*agent* en charge des actions, l'*agent* en charge de l'émulation va envoyer une requête pour obtenir l'objet `Building` précédemment créé (via le bouton '**Retreive Building Object**'). Une fois toutes les informations en main, on peut aisément choisir le capteur à simuler via l'arbre définissant le bâtiment et le tableau de sélection des capteurs (figure 5.16).

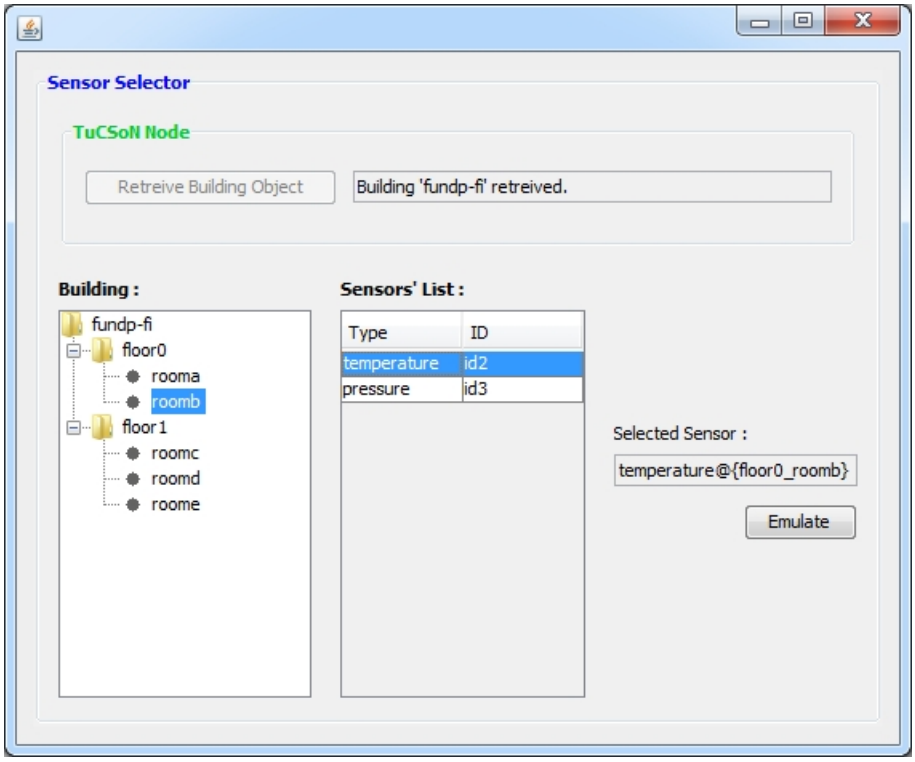

FIGURE 5.16 – SensorEmulator - Ecran principal

Le bouton '**Emulate**' permet de lancer une nouvelle fenêtre reprenant les informations du capteur à émuler (figure 5.17). L'envoi d'information peut être effectué de deux manières différentes :

- ***One single value*** : Envoi d'une seule valeur. Cette valeur peut être un nombre entier ou un chaîne de caractères.
- ***Random sequence*** : Envoi d'une séquence aléatoire (de nombres entiers). La période d'envoi doit être définie au préalable. Afin de fonctionner de manière cohérente avec l'entièreté du système, cette période doit être égale à la période d'échantillonnage des capteurs définies dans `EnvironmentManager.jar`.

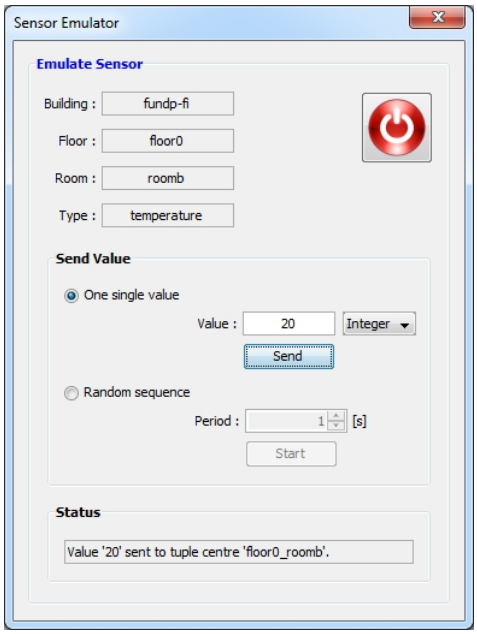

FIGURE 5.17 – SensorEmulator - Agent

5.7.2 Autres possibilités

L'environnement dans lequel le système a été créé se base sur le principe que des *tuples*, définis de manière structurée, alimentent un *tuple centre*. Des réactions sont implémentées dans ce *tuple centre* de sorte qu'il insère d'autres *tuples* dans d'autres *tuple centres* jusqu'à ce qu'une *agent* déclenche, sur l'arrivée d'un *tuple* spécifique, une action dans le 'monde réel'.

Pour donner du corps au développement dont il est question dans ce chapitre, on a calqué ce principe de fonctionnement sur la gestion de capteurs dans un bâtiment. Toutefois, un tel système peut être utilisé à d'autres fins. En effet, dans le cas d'une analyse syntaxique (détection de mots clés dans un texte, par exemple), on peut imaginer qu'un *agent* parcourt un document en envoyant chaque mot dans un *tuple centre*. Si ce *tuple centre* est programmé de sorte qu'il déclenche une action lors de l'arrivée d'un mot (règle simple), d'un ensemble de mots (règle composée) ou d'une séquence de mots (règle de type séquence), on obtient un système dont le but est tout à fait différent, mais dont les bases restent identiques à celles développées dans le cadre de ce mémoire.

5.8 Enregistrement des informations

Dans l'implémentation actuelle du système, chaque *tuple* arrivant dans un *tuple centre*, correspondant à l'étage du bâtiment et à la pièce où le capteur se situe, est enlevé après une période d'échantillonnage. L'enregistrement de ces informations en base de données n'a pas été implémentée ici, mais peut aisément être satisfait grâce à une réaction.

En effet, en définissant un *tuple centre* spécifique (`db_centre`, par exemple) dans lequel tous les *tuples* arrivant dans le système sont recopiés, on crée un espace de stockage temporaire pour les informations à enregistrer. En définissant un *agent* (**Java**) en charge de surveiller le contenu de ce *tuple centre*, on peut lui associer une requête vers une base de données locale (ou distante), dans laquelle il pourra enregistrer de manière structurée (c'est-à-dire, traduite du langage **TuCSoN**) toutes les informations entrantes. De là peut découler un système d'analyse de données plus poussé ou une représentation graphique à grande échelle.

La réaction décrite ci-après aiguille chaque *tuple* entrant vers le *tuple centre* `db_centre`. On y envoie aussi les informations concernant le *tuple centre* d'origine, ainsi qu'une information temporelle sur le noeud qui gère le système.

```
reaction(out(sensor_type(Id,Value)), (
  current_tuple(T),
    current_tc(N),
    current_time(Time),
    out_tc(db_centre, action(record(N,T,Time)))
     )).
```

5.9 Conclusion

Le présent développement est fonctionnel est permet de travailler avec un environement simple en y instaurant des règles qui suivent un gabarit prédéfini. Cependant, certaines fonctions restent encore à implémenter. Parmi celles-ci, on retrouve l'édition dynamique des éléments qui permettent la construction de l'environnement dans l'exécutable `EnvironmentManager.jar`, c'est-à-dire la gestion des boutons '**Edit Element**', '**Edit Sensor**', '**Edit Rule**' que l'on retrouve à la figure 5.1. Aussi, l'implémentation en **Java** des réactions relatives à la règle de type séquence n'est pas implémentée. Outre ces deux points importants, la gestion des exceptions et des erreurs de manipulation reste assez légère, mais suffisante pour un utilisateur averti.

L'implémentation, en l'état, donne toutefois une bonne représentation des fonctionnalités de bases de l'environnement **TuCSoN**. Les deux onglets de l'exécutable `EnvironmentManager.jar` donne à la fois une du vue côté utilisateur et une vue du côté développeur. On peut, de la sorte, vérifier la compréhension du système avec les traductions **ReSpecT** des réactions.

Une gestion de calendrier, au-dessus du système actuel, permettant d'implémenter les règles suivant les jours de la semaine, ou pour une plage horaire prédéfinie, aurait donné une application plus réaliste et dédié à la gestion (quotidienne) d'un bâtiment.

Les interfaces graphiques étant relativement bridés à l'utilsation de départ, à savoir, surveillance de capteurs dans un bâtiment, on ne sait pas dégager de suite des concepts génériques pouvant être réutiliser pour d'autres applications. Cependant, la construction des règles est assez détaillée pour porter leur implémentation vers un domaine relativement différent.

Chapitre 6

Conclusion

Le développement de l'application présentée dans ce rapport, même si elle n'utilise pas de réseau de capteurs sans fil comme exprimé dans le sujet, m'a permis d'accroître mes connaissances dans les systèmes d'information en général, et plus particulièrement, en ce qui concerne les langages de coordination. En effet, les délais trop importants annoncés pour l'obtention des capteurs m'ont poussé à m'orienter plus en profondeur dans les recherches et la documentation relatives à l'environnement TuCSoN et au langage ReSpecT. Deux sujets qui m'étaient alors inconnus.

L'interprétation des règles et leur traduction en langage "machine" montre jusqu'ou on peut aller dans le développement de telles applications. Il est à noter que pour le travail dont il est question ici, la partie intelligente est entièrement située dans l'espace de coordination de TuCSoN. Le programme Java, quant à lui, est en charge des interfaces graphiques et de l'interprétation des règles entrées par l'utilisateur.

Le système final tel que présenté dans la section précédente est fonctionnel malgré le manque de certaines fonctionnalités. Il pourra sans doute servir de base pour un travail future, ou de point de comparaison dans le cas d'une refonte du système.

Si je devais refaire le même projet, je commencerais par changer la gestion du temps qui m'est imparti. Aussi, je pense qu'une mise à plat complète des spécifications fonctionnelles auraient permis de facilité le développement et de la structurer différemment. Toutefois, le résultat obtenu reste au-delà de mes espérances au vu du peu de bagages dont je disposais dans le domaine.

D'un point de vue personnel, ce travail, comme les études qui ont précédé, m'ont apporté une autre vision du domaine des systèmes d'information au sens large du terme. Venant d'une formation en électronique, je peux maintenant avoir une vue globale sur un système complet, *hardware* et *software*, en apportant les critiques constructives qui font d'un ingénieur ce qu'il est.

Bibliographie

[1] aliCE Research Group. Agents, Languages & Infrastructure for Complexity Engineering. `www.alice.unibo.it`.

[2] WiFi Alliance. WiFi Certified makes it WiFi. `www.wi-fi.org`.

[3] ZigBee Alliance. ZigBee, Control your world. `www.zigbee.org`.

[4] Bluetooth Special Interest Group. Bluetooth. `www.bluetooth.org`.

[5] IrDA Special Interest Group. IrDA, The secure wireless link. `www.irda.org`.

[6] A Division of Dynastream Innovations Inc. ANT, The power of less. `www.thisisant.com`.

[7] Andrea Omicini. Towards a notion of agent coordination context. In Dan C. Marinescu and Craig Lee, editors, *Process Coordination and Ubiquitous Computing*, chapter 12, pages 187–200. CRC Press, Boca Raton, FL, USA, octobre 2002.

[8] Andrea Omicini and Enrico Denti. Formal ReSpecT. *Electronic Notes in Theoretical Computer Science*, 48 :179–196, juin 2001. Declarative Programming – Selected Papers from AGP 2000, La Habana, Cuba, 4–6 December 2000.

[9] Andrea Omicini and Enrico Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3) :277–294, 2001.

[10] Andrea Omicini and George A. Papadopoulos. Why coordination models and languages in AI ? *Applied Artificial Intelligence*, 15(1) :1–10, janvier 2001. Special Issue : Coordination Models and Languages in AI.

[11] Alessandro Ricci. *TuCSoN Guide*. aliCE Research Group, 2006.

[12] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. Agent coordination context : From theory to practice. In Robert Trappl, editor, *Cybernetics and Systems 2004*, volume 2, pages 618–623, Vienna, Austria, 13–16 avril 2004. Austrian Society for Cybernetic Studies. 17th European Meeting on Cybernetics and Systems Research (EMCSR 2004), 4th International Symposium "From Agent Theory to Theory Implementation (AT2AI-4)". Proceedings.

[13] Phill Smith. Comparisons between Low Power Wireless Technologies, mai 2011. `www.csr.com` - Whitepaper.

Annexe A

readme.txt

```
##############################################
#          FUNDP - 2011-2012                 #
#   Remy Woine - Thesis Application          #
##############################################

This folder contains the three NetBeans Projects :
- EnvironmentManager
- ActionManager
- SensorEmulator

The three relative .jar files are in the 'dist' repository of each project.
However, three .bat files can launch each application. Those are :
- LaunchEnvironmentManager.bat
- LaunchActionManager.bat
- LaunchSensorEmulator.bat

Before attempting any requests between the application, ensure that the TuCSoN is
    actually launched.
This can be achieved in the EnvironmentManager application, in the second tab with
    the button names 'Launch TuCSoN Node'.

The other .bat files are only used to interact with the TuCSoN API.
They are not necessary to the application;
- LaunchNode.bat launches the TuCSoN Node (it has to be done before launching the
    other .bat files).
- LaunchInspector.bat intends to inspect tuples, reactions, specifications in the
    TuCSoN coordination space.
- LaunchCLIAgent.bat starts an agent ready to interact with the TuCSoN coordination
     space.
```

Listing A.1 – README

Annexe B

package be.woine.thesis.building.common

```java
package be.woine.thesis.building.common;

import java.io.Serializable;
import java.util.ArrayList;

/**
 * This class defines a Building object.
 * @author Remy
 * @version 1.0
 */
public class Building implements Serializable {

    private String name = "";
    private ArrayList<Floor> floorList = new ArrayList<Floor>();

    /**
     * Empty constructor of the Building object.
     */
    public Building() {
    }

    /**
     * Constructor of the Building object.
     * @param name Name of the building.
     */
    public Building(String name) {
        this.name = name;
    }

    /**
     * Constructor of the Building object.
     * @param name Name of the building.
     * @param floorList The list of the floors present in the building.
     */
    public Building(String name, ArrayList<Floor> floorList) {
        this.name = name;
        this.floorList = floorList;
    }

    /**
     * Gets the building's name.
     * @return The name of the bulding.
     */
    public String getName() {
        return name;
    }

    /**
```

```
     * Sets the bulding's name.
     * @param name The name of the building.
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Gets the floors'list present in the building.
     * @return The floors'list present in the building.
     */
    public ArrayList<Floor> getFloorList() {
        return floorList;
    }

    /**
     * Sets the floors'list in the building.
     * @param floorList The floors'list in the building.
     */
    public void setFloorList(ArrayList<Floor> floorList) {
        this.floorList = floorList;
    }

    /**
     * Override of the toString() method to get the name of the building.
     * @return The name of the building.
     */
    @Override
    public String toString() {
        return this.name;
    }

    /**
     * Gets the floor at the mentionned index.
     * @param index Index of the floor to get.
     * @return The floor at the mentionned index.
     */
    public Floor getFloor(int index) {
        return this.floorList.get(index);
    }

    /**
     * Adds a floor in the building
     * @param floor The floor to add in the building.
     */
    public void addFloor(Floor floor) {
        this.floorList.add(floor);
    }

    /**
     * Gets the number of floors within a building.
     * @return The number of floors within a building.
     */
    public int getFloorCount() {
        return this.floorList.size();
    }

}
```

Listing B.1 – Building Class

```
package be.woine.thesis.building.common;

import java.io.Serializable;
import java.util.ArrayList;

/**
 * This class defines the Floor object.
 * @author Remy
```

```
 * @version 1.0
 */
public class Floor implements Serializable {

    private String name = "";
    private ArrayList<Room> roomList = new ArrayList<Room>();

    /**
     * Empty constructor of the floor object.
     */
    public Floor() {
    }

    /**
     * Constructor of the floor object.
     * @param name Name of the floor.
     */
    public Floor(String name) {
        this.name = name;
    }

    /**
     * Constructor of the floor object.
     * @param name Name of the floor.
     * @param roomList List of the rooms present on the floor.
     */
    public Floor(String name, ArrayList<Room> roomList) {
        this.name = name;
        this.roomList = roomList;
    }

    /**
     * Gets the floor's name.
     * @return The name of the floor.
     */
    public String getName() {
        return name;
    }

    /**
     * Sets the floor's name.
     * @param name The name of the floor.
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Gets the rooms'list present on the floor.
     * @return The rooms'list present on the floor.
     */
    public ArrayList<Room> getRoomList() {
        return roomList;
    }

    /**
     * Sets the rooms'list on the floor.
     * @param roomList The room's list on the floor.
     */
    public void setRoomList(ArrayList<Room> roomList) {
        this.roomList = roomList;
    }

    /**
     * Override of the toString() method to get the name of the floor.
     * @return The name of the floor.
     */
    @Override
    public String toString() {
        return this.name;
```

```
    }

    /**
     * Gets the room at the mentionned index.
     * @param index The index of the room to get.
     * @return The room at the selected index.
     */
    public Room getRoom(int index) {
        return this.roomList.get(index);
    }

    /**
     * Adds the room on the floor.
     * @param room The room to add on the floor.
     */
    public void addRoom(Room room) {
        this.roomList.add(room);
    }

    /**
     * Gets the number of rooms on a floor.
     * @return The number of rooms on a floor.
     */
    public int getRoomCount() {
        return this.roomList.size();
    }

}
```

Listing B.2 – Floor Class

```
package be.woine.thesis.building.common;

import java.io.Serializable;
import java.util.ArrayList;

/**
 * This class defines the Room object.
 * @author Remy
 * @version 1.0
 */
public class Room implements Serializable {

    private String name = "";
    private ArrayList<Sensor> sensorList = new ArrayList<Sensor>();

    /**
     * Empty constructor of the Room object.
     */
    public Room() {
    }

    /**
     * Constructor of the Room object.
     * @param name Name of the room.
     */
    public Room(String name) {
        this.name = name;
    }

    /**
     * Constructor of the Room object.
     * @param name Name of the room.
     * @param sensorList List of sensors present in the room.
     */
    public Room(String name, ArrayList<Sensor> sensorList) {
        this.name = name;
        this.sensorList = sensorList;
    }
```

```
    /**
     * Gets the room's name.
     * @return The name of the room.
     */
    public String getName() {
        return name;
    }

    /**
     * Sets the room's name.
     * @param name The name of the room.
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Gets the sensors'list present in the room.
     * @return The sensors'list present in the room.
     */
    public ArrayList<Sensor> getSensorList() {
        return sensorList;
    }

    /**
     * Sets the sensors'list in the room.
     * @param sensorList The sensors'list in the room.
     */
    public void setSensorList(ArrayList<Sensor> sensorList) {
        this.sensorList = sensorList;
    }

    /**
     * Override of the toString() method to get the name of the room.
     * @return The name of the room.
     */
    @Override
    public String toString() {
        return this.name;
    }

    /**
     * Gets the sensor at the mentionned index.
     * @param index Index of the sensor to get.
     * @return The sensor at the mentionned index.
     */
    public Sensor getSensor(int index) {
        return this.sensorList.get(index);
    }

    /**
     * Adds a sensor in the room.
     * @param sensor Sensor to add in the room.
     */
    public void addSensor(Sensor sensor) {
        this.sensorList.add(sensor);
    }

    /**
     * Gets the number of sensors within a room.
     * @return The number of sensors within a room.
     */
    public int getSensorCount() {
        return this.sensorList.size();
    }

}
```

Listing B.3 – Room Class

```java
package be.woine.thesis.building.common;

import java.io.Serializable;

/**
 * This class defines a Sensor object.
 * @author Remy
 * @version 1.0
 */
public class Sensor implements Serializable {

    private String id = "";
    private SensorType type = SensorType.UNDEFINED;

    /**
     * Empty constructor for the Sensor object.
     */
    public Sensor() {
    }

    /**
     * Constructor of the Sensor object.
     * @param id Unique identifier of the sensor.
     * @param type Type of the sensor defining by SensorType.
     */
    public Sensor(String id, SensorType type) {
        this.id = id;
        this.type = type;
    }

    /**
     * Gets the sensor identifier.
     * @return The sensor's unique identifier.
     */
    public String getId() {
        return id;
    }

    /**
     * Sets the sensor identifier.
     * @param id The sensor's unique identifier.
     */
    public void setId(String id) {
        this.id = id;
    }

    /**
     * Gets the sensor's type.
     * @return The sensor's type defining by a SensorType value.
     */
    public SensorType getType() {
        return type;
    }

    /**
     * Sets the sensor's type.
     * @param type The sensor's type defining by a SensorType value.
     */
    public void setType(SensorType type) {
        this.type = type;
    }

    /**
     * Override of the toString() method to get a correct string representing the
       sensor
     * @return A string representing the sensor's attributes.
     */
    @Override
    public String toString() {
```

```
        return "Sensor{" + "id=" + id + ", type=" + type + "}";
    }

}
```

Listing B.4 – Sensor Class

```
package be.woine.thesis.building.common;

import java.io.Serializable;

/**
 * This class enumerates the possible sensor's type used in the application.
 * @author Remy
 * @version 1.0
 */
public class SensorType implements Serializable {

    private int value;

    private static final int _UNDEFINED = -1;
    public static final SensorType UNDEFINED = new SensorType(_UNDEFINED);
    private static final int _TEMPERATURE = 0;
    public static final SensorType TEMPERATURE = new SensorType(_TEMPERATURE);
    private static final int _HUMIDITY = 1;
    public static final SensorType HUMIDITY = new SensorType(_HUMIDITY);
    private static final int _LUMINOSITY = 2;
    public static final SensorType LUMINOSITY = new SensorType(_LUMINOSITY);
    private static final int _PRESSURE = 3;
    public static final SensorType PRESSURE = new SensorType(_PRESSURE);

    /**
     * Constructor of the sensor's type.
     * @param value An integer value representing the sensor's type.
     */
    private SensorType(int value) {
        this.value = value;
    }

    /**
     * Override of the toString() method to get a correct string representing the
       sensor's type.
     * @return A String representing the sensor's type.
     */
    @Override
    public String toString() {
        if(this.value == _UNDEFINED)
            return "undefined";
        else if(this.value == _TEMPERATURE)
            return "temperature";
        else if(this.value == _HUMIDITY)
            return "humidity";
        else if(this.value == _LUMINOSITY)
            return "luminosity";
        else if(this.value == _PRESSURE)
            return "pressure";
        else
            return "";
    }

    /**
     * Gets a single lettre used as a variable for the further rules involving this
       sensor type.
     * @return A single lettre representing the sensor type's variable.
     */
    public String getVar() {
        if(this.value == _TEMPERATURE)
            return "T";
        else if(this.value == _HUMIDITY)
```

```
            return "H";
        else if(this.value == _LUMINOSITY)
            return "L";
        else if(this.value == _PRESSURE)
            return "P";
        else
            return "";
    }

}
```

Listing B.5 – SensorType Class

Annexe C

package be.woine.thesis.rule.common

```java
package be.woine.thesis.rule.common;

import be.woine.thesis.action.common.Action;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * This class defines the rule at a higher level.
 * @author Remy
 * @version 1.0
 */
public class Rule implements Serializable {

    private int id;
    private static final AtomicInteger atomicId = new AtomicInteger();
    private Action action;
    private ArrayList<CompoundRule> compoundRuleList;

    /**
     * Empty constructor for the Rule object.
     */
    public Rule() {
        this.id = atomicId.incrementAndGet();
        this.compoundRuleList = new ArrayList<CompoundRule>();
        this.action = new Action();
    }

    /**
     * Constructor of the Rule object.
     * @param compoundRuleList The compound rule's list.
     */
    public Rule(ArrayList<CompoundRule> compoundRuleList, Action action) {
        this.id = atomicId.incrementAndGet();
        this.compoundRuleList = compoundRuleList;
        this.action = action;
    }

    /**
     * Gets the type of the rule, depending og its constitution.
     * @return The type of the rule.
     */
    public RuleType getType() {
        if(this.compoundRuleList.size() == 1) {
            return this.compoundRuleList.get(0).getType();
        } else if(this.compoundRuleList.size() > 1) {
            return RuleType.SEQUENCE;
        } else
            return RuleType.UNDEFINED;
    }
```

```
/**
 * Gets the action triggered by the rule satisfaction.
 * @return The action triggered by the rule.
 */
public Action getAction() {
    return this.action;
}

/**
 * Sets the action to be triggered by the rule satisfaction.
 * @param action The action to be triggered by the rule.
 */
public void setAction(Action action) {
    this.action = action;
}

/**
 * Gets the compound rule's list of the rule.
 * @return The compound rule's list.
 */
public ArrayList<CompoundRule> getCompoundRuleList() {
    return this.compoundRuleList;
}

/**
 * Sets the compound rule's list of the rule.
 * @param compoundRuleList The compound rule's list to set.
 */
public void setCompoundRuleList(ArrayList<CompoundRule> compoundRuleList) {
    this.compoundRuleList = compoundRuleList;
}

/**
 * Adds a compound rule to the compound rule's list of the rule.
 * @param compoundRule The compound rule to add.
 */
public void addCompoundRule(CompoundRule compoundRule) {
    this.compoundRuleList.add(compoundRule);
}

/**
 * Adds a compound rule to the compound rule's list of the rule at the
     mentionned index.
 * @param compoundRule The compound rule to add.
 * @param index The index at which the compound rule should be added.
 */
public void addCompoundRule(CompoundRule compoundRule, int index) {
    this.compoundRuleList.add(index, compoundRule);
}

/**
 * Gets the compound rule from the compound rule's list at the mentionned index
     .
 * @param index Index from of the compound rule to get.
 * @return The compound rule at the mentionned index.
 */
public CompoundRule getCompoundRule(int index) {
    return this.compoundRuleList.get(index);
}

/**
 * Gets the Compound Rule unique identifier.
 * @return An integer representing the rule identifier.
 */
public int getId() {
    return this.id;
}

/**
```

```
     * Override of the toString() method to get a correct string representation
       according to the rule's type.
     * @return A string representing the rule.
     */
    @Override
    public String toString() {

        String str = "";

        for(int i = 0; i < this.compoundRuleList.size(); i++) {

            str += this.compoundRuleList.get(i).toString();

            if(i != (this.compoundRuleList.size() - 1))
                str += " -> ";
        }

        return str;

    }

}
```

Listing C.1 – Rule Class

```
package be.woine.thesis.rule.common;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * This class defines a compound rule, i.e. a rule composed by several simple rules
    .
 * @author Remy
 * @version 1.0
 */
public class CompoundRule implements Serializable {

    private int id;
    private static final AtomicInteger atomicId = new AtomicInteger();
    private ArrayList<SimpleRule> simpleRuleList;

    /**
     * Empty constructor for the CompoundRule object.
     */
    public CompoundRule() {
        this.id = atomicId.incrementAndGet();
        this.simpleRuleList = new ArrayList<SimpleRule>();
    }

    /**
     * Constructor of the CompoundRule object.
     * @param simpleRuleList List of the simple rule that built the CompoundRule.
     */
    public CompoundRule(ArrayList<SimpleRule> simpleRuleList) {
        this.id = atomicId.incrementAndGet();
        this.simpleRuleList = simpleRuleList;
    }

    /**
     * Gets the type of the rule, depending og its constitution.
     * @return The type of the rule.
     */
    public RuleType getType() {
        if(this.simpleRuleList.size() == 1) {
            return this.simpleRuleList.get(0).getType();
        } else if(this.simpleRuleList.size() > 1) {
            return RuleType.COMPOUND;
```

```
        } else
            return RuleType.UNDEFINED;
    }

    /**
     * Gets the simple rule's list.
     * @return The simple rule's list.
     */
    public ArrayList<SimpleRule> getSimpleRuleList() {
        return this.simpleRuleList;
    }

    /**
     * Set the simple rule's list.
     * @param simpleRuleList The simple rule's list to set.
     */
    public void setSimpleRuleList(ArrayList<SimpleRule> simpleRuleList) {
        this.simpleRuleList = simpleRuleList;
    }

    /**
     * Adds a simple rule to the simple rule's list.
     * @param simpleRule The simple rule to add.
     */
    public void addSimpleRule(SimpleRule simpleRule) {
        this.simpleRuleList.add(simpleRule);
    }

    /**
     * Gets a simple rule from the list at the mentionned index.
     * @param index Index ofthe simple rule to get.
     * @return The simple rule at the mentionned index.
     */
    public SimpleRule getSimpleRule(int index) {
        return this.simpleRuleList.get(index);
    }

    /**
     * Gets the Compound Rule unique identifier.
     * @return An integer representing the rule identifier.
     */
    public int getId() {
        return this.id;
    }

    /**
     * Override of the toString() method to get a correct string representation
       according to the rule's type.
     * @return A string representing the rule.
     */
    @Override
    public String toString() {

        String str = "";

        for(int i = 0; i < this.simpleRuleList.size(); i++) {

            str += this.simpleRuleList.get(i).toString();

            if(i != (this.simpleRuleList.size() - 1))
                str += " AND ";

        }

        return str;

    }

}
```

Listing C.2 – CompoundRule Class

```
package be.woine.thesis.rule.common;

import java.io.Serializable;
import java.util.ArrayList;

/**
 * This class defines a simple rule.
 * @author Remy
 * @version 1.0
 */
public class SimpleRule implements Serializable {

    private RuleOperator specialOperator = RuleOperator.UNDEFINED;
    private ValueSource source;
    private ArrayList<AbstractRuleBody> ruleBodyList;

    /**
     * Empty constructor of the SimpleRule object.
     */
    public SimpleRule() {
        this.source = null;
        this.ruleBodyList = new ArrayList<AbstractRuleBody>();
    }

    /**
     * Constructor for the SimpleRule object.
     * @param source Source of the value involved into the rule.
     * @param ruleBodyList List of the rule's body.
     */
    public SimpleRule(ValueSource source, ArrayList<AbstractRuleBody> ruleBodyList)
         {
        this.source = source;
        this.ruleBodyList = ruleBodyList;
    }

    /**
     * Gets the type of the rule according to the number of the rule's body.
     * @return The type of the rule amongst FIXED and RANGE.
     */
    public RuleType getType() {
        if(this.ruleBodyList.size() == 1)
            return RuleType.FIXED;
        else if(this.ruleBodyList.size() == 2)
            return RuleType.RANGE;
        else
            return RuleType.UNDEFINED;
    }

    /**
     * Gets the list of the rule's body.
     * @return The list of the rule's body.
     */
    public ArrayList<AbstractRuleBody> getRuleBodyList() {
        return this.ruleBodyList;
    }

    /**
     * Sets the list of the rule's body.
     * @param ruleBodyList The list of the rule's body.
     */
    public void setRuleBodyList(ArrayList<AbstractRuleBody> ruleBodyList) {
        this.ruleBodyList = ruleBodyList;
    }

    /**
```

```
     * Gets the source of the value involved into the rule.
     * @return The source of the value.
     */
    public ValueSource getSource() {
        return this.source;
    }

    /**
     * Sets the source of the value to be involved into the rule.
     * @param source The source of the value.
     */
    public void setSource(ValueSource source) {
        this.source = source;
    }

    /**
     * Gets the special operator in case of a RANGE rule type.
     * @return The special rule operator.
     */
    public RuleOperator getSpecialOperator() {
        return this.specialOperator;
    }

    /**
     * Sets the special operator in cas of a RANGE rule type.
     * @param specialOperator The special operator.
     */
    public void setSpecialOperator(RuleOperator specialOperator) {
        this.specialOperator = specialOperator;
    }

    /**
     * Adds a rule body to the list of the rule's body.
     * @param ruleBody The rule's body to add.
     */
    public void addRuleBody(AbstractRuleBody ruleBody) {
        this.ruleBodyList.add(ruleBody);
    }

    /**
     * Gets the rule's body at the mentionned index.
     * @param index The index of the rule's body to get.
     * @return The rule's body at the mentionned index.
     */
    public AbstractRuleBody getRuleBody(int index) {
        return this.ruleBodyList.get(index);
    }

    /**
     * Overrides the toString() method to get a correct string according to the
         rule's type.
     * @return A string representing the rule.
     */
    @Override
    public String toString() {

        String str = "";

        if(this.getType() != RuleType.UNDEFINED) {
            str += "(";
            str += this.source.building.getFloor(this.source.floorIndex).getRoom(
                this.source.roomIndex).getSensor(this.source.sensorIndex).getType()
                .getVar();
            str += "@";
            str += this.source.building.getFloor(this.source.floorIndex).getName();
            str += "_";
            str += this.source.building.getFloor(this.source.floorIndex).getRoom(
                this.source.roomIndex).getName();
            str += ")";
        }
```

```
        if(this.getType() == RuleType.FIXED) {
            str += fixedRuleToString();
        } else if(this.getType() == RuleType.RANGE) {
            str += rangeRuleToString();
        }

        return str;

    }


    private String fixedRuleToString(){

        String str = "";

        str = this.ruleBodyList.get(0).operator.getSymbol() + this.ruleBodyList.get
            (0).value;

        return str;
    }

    private String rangeRuleToString(){

        String str = "";

        str = this.specialOperator.getSymbol();

        if(this.ruleBodyList.get(0).operator.toString().compareTo(RuleOperator.
            GREATER_THAN.toString()) == 0) {
            str += "]";
        } else if(this.ruleBodyList.get(0).operator.toString().compareTo(
            RuleOperator.GREATER_THAN_OR_EQUAL_TO.toString()) == 0) {
            str += "[";
        }

        str += this.ruleBodyList.get(0).value;
        str += ":";
        str += this.ruleBodyList.get(1).value;

        if(this.ruleBodyList.get(1).operator.toString().compareTo(RuleOperator.
            SMALLER_THAN.toString()) == 0) {
            str += "[";
        } else if(this.ruleBodyList.get(1).operator.toString().compareTo(
            RuleOperator.SMALLER_THAN_OR_EQUAL_TO.toString()) == 0) {
            str += "]";
        }

        return str;
    }

}
```

Listing C.3 – SimpleRule Class

```
package be.woine.thesis.rule.common;

import java.io.Serializable;

/**
 * This class defines a generic RuleBody used for simple rules construction.
 * @author Remy
 * @version 1.0
 */
public abstract class AbstractRuleBody<T> implements Serializable {

    protected T value;
    protected RuleOperator operator;
```

```
    /**
     * Gets the operator of the rule's body.
     * @return The operator of the rule's body.
     */
    public RuleOperator getOperator() {
        return operator;
    }

    /**
     * Sets the operator of the rule's body.
     * @param operator The opertor of the rule's body.
     */
    public void setOperator(RuleOperator operator) {
        this.operator = operator;
    }

    /**
     * Gets the value which is involved within the rule.
     * @return The value involved within the rule.
     */
    public T getValue() {
        return value;
    }

    /**
     * Sets the value to be involved within the rule.
     * @param value The value to be involved within the rule.
     */
    public void setValue(T value) {
        this.value = value;
    }

}
```

Listing C.4 – AbstractRuleBody Class

```
package be.woine.thesis.rule.common;

/**
 * This class defines the Integer implementation of the RuleBody.
 * @author Remy
 * @version 1.0
 */
public class IntegerRuleBody extends AbstractRuleBody<Integer> {

    /**
     * Constructor of the IntegerRuleBody object.
     * @param value The integer value of the RuleBody.
     * @param operator The operator used with the value.
     */
    public IntegerRuleBody(Integer value, RuleOperator operator) {
        super.value = value;
        super.operator = operator;
    }

}
```

Listing C.5 – IntegerRuleBody Class

```
package be.woine.thesis.rule.common;

/**
 * This class defines the String implementation of the RuleBody.
 * @author Remy
 * @version 1.0
 */
public class StringRuleBody extends AbstractRuleBody<String> {
```

```
    /**
     * Constructor of the StringRuleBody object.
     * @param value The string value of the RuleBody.
     * @param operator The operator used with the value.
     */
    public StringRuleBody(String value, RuleOperator operator) {
        super.value = "'" + value + "'";
        super.operator = operator;
    }

}
```

Listing C.6 – StringRuleBody Class

```
package be.woine.thesis.rule.common;

import java.io.Serializable;

/**
 * This class define the different types a rule can take.
 * @author Remy
 * @version 1.0
 */
public class RuleType implements Serializable {

    private int value;

    private static final int _UNDEFINED = -1;
    public static final RuleType UNDEFINED = new RuleType(_UNDEFINED);
    private static final int _FIXED = 0;
    public static final RuleType FIXED = new RuleType(_FIXED);
    private static final int _RANGE = 1;
    public static final RuleType RANGE = new RuleType(_RANGE);
    private static final int _COMPOUND = 2;
    public static final RuleType COMPOUND = new RuleType(_COMPOUND);
    private static final int _SEQUENCE = 3;
    public static final RuleType SEQUENCE = new RuleType(_SEQUENCE);

    /**
     * Constructor of the rule's type.
     * @param value An integer value representing the rule's type.
     */
    private RuleType(int value) {
        this.value = value;
    }

    /**
     * Override of the toString() method to get a correct string representing the
       rule's type.
     * @return A String representing the rule's type.
     */
    @Override
    public String toString() {
        if(this.value == _UNDEFINED)
            return "undefined";
        else if(this.value == _FIXED)
            return "fixed";
        else if(this.value == _RANGE)
            return "range";
        else if(this.value == _COMPOUND)
            return "compound";
        else if(this.value == _SEQUENCE)
            return "sequence";
        else
            return "";
    }

}
```

Listing C.7 – RuleType Class

```
package be.woine.thesis.rule.common;

import java.io.Serializable;

/**
 * This class defines the different operators used for a rule.
 * @author Remy
 * @version 1.0
 */
public class RuleOperator implements Serializable {

    private int value;

    private static final int _UNDEFINED = -1;
    public static final RuleOperator UNDEFINED = new RuleOperator(_UNDEFINED);
    private static final int _SMALLER_THAN = 0;
    public static final RuleOperator SMALLER_THAN = new RuleOperator(_SMALLER_THAN)
        ;
    private static final int _SMALLER_THAN_OR_EQUAL_TO = 1;
    public static final RuleOperator SMALLER_THAN_OR_EQUAL_TO = new RuleOperator(
        _SMALLER_THAN_OR_EQUAL_TO);
    private static final int _GREATER_THAN = 2;
    public static final RuleOperator GREATER_THAN = new RuleOperator(_GREATER_THAN)
        ;
    private static final int _GREATER_THAN_OR_EQUAL_TO = 3;
    public static final RuleOperator GREATER_THAN_OR_EQUAL_TO = new RuleOperator(
        _GREATER_THAN_OR_EQUAL_TO);
    private static final int _EQUAL_TO = 4;
    public static final RuleOperator EQUAL_TO = new RuleOperator(_EQUAL_TO);
    private static final int _NOT_EQUAL_TO = 5;
    public static final RuleOperator NOT_EQUAL_TO = new RuleOperator(_NOT_EQUAL_TO)
        ;
    private static final int _INCLUDED_IN = 6;
    public static final RuleOperator INCLUDED_IN = new RuleOperator(_INCLUDED_IN);
    private static final int _NOT_INCLUDED_IN = 7;
    public static final RuleOperator NOT_INCLUDED_IN = new RuleOperator(
        _NOT_INCLUDED_IN);

    /**
     * Constructor of the rule's operator.
     * @param value An integer value representing the rule's operator.
     */
    public RuleOperator(int value) {
        this.value = value;
    }

    /**
     * Override of the toString() method to get a correct string representing the
         rule's operator.
     * @return A String representing the rule's operator.
     */
    @Override
    public String toString() {
        if(this.value == _UNDEFINED)
            return "undefined";
        else if(this.value == _SMALLER_THAN)
            return "smaller";
        else if(this.value == _SMALLER_THAN_OR_EQUAL_TO)
            return "smaller_equal";
        else if(this.value == _GREATER_THAN)
            return "greater";
        else if(this.value == _GREATER_THAN_OR_EQUAL_TO)
            return "greater_equal";
        else if(this.value == _EQUAL_TO)
            return "equal";
```

```
        else if(this.value == _NOT_EQUAL_TO)
            return "not_equal";
        else if(this.value == _INCLUDED_IN)
            return "included";
        else if(this.value == _NOT_INCLUDED_IN)
            return "not_included";
        else
            return "";
    }

    /**
     * Gets the description of the rule operator.
     * @return A string describing the rule operator.
     */
    public String getDescription() {
        if(this.value == _UNDEFINED)
            return "undefined";
        else if(this.value == _SMALLER_THAN)
            return "is smaller than";
        else if(this.value == _SMALLER_THAN_OR_EQUAL_TO)
            return "is smaller than or equal to";
        else if(this.value == _GREATER_THAN)
            return "is greater than";
        else if(this.value == _GREATER_THAN_OR_EQUAL_TO)
            return "is greater or equal to";
        else if(this.value == _EQUAL_TO)
            return "is equal to";
        else if(this.value == _NOT_EQUAL_TO)
            return "is not equal to";
        else if(this.value == _INCLUDED_IN)
            return "is included in";
        else if(this.value == _NOT_INCLUDED_IN)
            return "is not included in";
        else
            return "";
    }

    /**
     * Gets the symbol of the rule's operator.
     * @return A String representing the symbol of the rule's operator.
     */
    public String getSymbol() {
        if(this.value == _SMALLER_THAN)
            return "<";
        else if(this.value == _SMALLER_THAN_OR_EQUAL_TO)
            return "<=";
        else if(this.value == _GREATER_THAN)
            return ">";
        else if(this.value == _GREATER_THAN_OR_EQUAL_TO)
            return ">=";
        else if(this.value == _EQUAL_TO)
            return "=";
        else if(this.value == _NOT_EQUAL_TO)
            return "/=";
        else if(this.value == _INCLUDED_IN)
            return "inc";
        else if(this.value == _NOT_INCLUDED_IN)
            return "!inc";
        else
            return "";
    }

    /**
     * Gets the rule's operator to be used for mathematical expressions.
     * @return A String representing the rule's operator.
     */
    public String getOperator() {
        if(this.value == _SMALLER_THAN)
            return "<";
        else if(this.value == _SMALLER_THAN_OR_EQUAL_TO)
```

```
            return "=<";
        else if(this.value == _GREATER_THAN)
            return ">";
        else if(this.value == _GREATER_THAN_OR_EQUAL_TO)
            return ">=";
        else if(this.value == _EQUAL_TO)
            return "=";
        else if(this.value == _NOT_EQUAL_TO)
            return "\\=";
        else
            return "";
    }

}
```

Listing C.8 – RuleOperator Class

```
package be.woine.thesis.rule.common;

import be.woine.thesis.building.common.Building;
import java.io.Serializable;

/**
 * This class defines the source of the value involved into the rule.
 * @author Remy
 * @version 1.0
 */
public class ValueSource implements Serializable {

    public Building building;
    public int floorIndex;
    public int roomIndex;
    public int sensorIndex;

    /**
     * Constructor of the ValueSource object.
     * @param building Building from which the value is issued.
     * @param floorIndex Index of the floor in the building.
     * @param roomIndex Index of the room on the floor.
     * @param sensorIndex index of the sensor in the room.
     */
    public ValueSource(Building building, int floorIndex, int roomIndex, int
        sensorIndex) {
        this.building = building;
        this.floorIndex = floorIndex;
        this.roomIndex = roomIndex;
        this.sensorIndex = sensorIndex;
    }

}
```

Listing C.9 – ValueSource Class

Annexe D

package be.woine.thesis.action.common

```java
package be.woine.thesis.action.common;

import java.io.Serializable;


/**
 * This class defines an action that should be triggered when the rule is verified.
 * @author Remy
 * @version 1.0
 */
public class Action implements Serializable {

    private ActionType type = ActionType.UNDEFINED;
    private String name;

    /**
     * Empty constructor of the Action object.
     */
    public Action() {
        this.name = "";
    }

    /**
     * Constructor of the Action object.
     * @param name The name of the action.
     */
    public Action(String name) {
        this.name = name;
    }

    /**
     * Constructor of the Action object.
     * @param type The type of the action.
     */
    public Action(ActionType type) {
        this.type = type;
        this.name = type.toString();
    }

    /**
     * Gets the name of the action.
     * @return The name of the action.
     */
    public String getName() {
        return name;
    }

    /**
```

```
     * Sets the name of the action.
     * @param name The name of the action.
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Gets the type of the action.
     * @return The type of the action.
     */
    public ActionType getType() {
        return type;
    }

    /**
     * Sets the type of the action.
     * @param type The type of the action.
     */
    public void setType(ActionType type) {
        this.type = type;
    }

    /**
     * Override of the toString() method to get a correct string representing the
         action.
     * @return The name of the action.
     */
    @Override
    public String toString() {
        return this.name;
    }

}
```

Listing D.1 – Action Class

```
package be.woine.thesis.action.common;

import java.io.Serializable;

/**
 * This class define the different types an action can take.
 * @author Remy
 * @version 1.0
 */
public class ActionType implements Serializable {

    private int value;

    private static final int _UNDEFINED = -1;
    public static final ActionType UNDEFINED = new ActionType(_UNDEFINED);
    private static final int _PRINT_CONSOLE = 0;
    public static final ActionType PRINT_CONSOLE = new ActionType(_PRINT_CONSOLE);
    private static final int _DIALOG_ALERT = 1;
    public static final ActionType DIALOG_ALERT = new ActionType(_DIALOG_ALERT);

    /**
     * Constructor of the rule's type.
     * @param value An integer value representing the action's type.
     */
    private ActionType(int value) {
        this.value = value;
    }

    /**
     * Override of the toString() method to get a correct string representing the
         action's type.
     * @return A String representing the action's type.
```

```
     */
    @Override
    public String toString() {
        if(this.value == _UNDEFINED)
            return "undefined";
        else if(this.value == _PRINT_CONSOLE)
            return "print_console";
        else if(this.value == _DIALOG_ALERT)
            return "dialog_alert";
        else
            return "";
    }

}
```

Listing D.2 – ActionType Class

Annexe E

package be.woine.thesis.agent

```
package be.woine.thesis.agent;

import alice.logictuple.*;
import alice.tucson.api.*;
import java.util.Date;
import javax.swing.JTextArea;

/**
 * This class defines the monitor agent in charge of monitoring all the incoming
     tuples
 * @author Remy
 * @version 1.0
 */
public class MonitorAgent extends Thread {

    private String name;
    private JTextArea jTextAreaConsole;
    private Boolean running = true;

    private TucsonContextSynch ctx;

    /**
     * Constructor of the monitor agent.
     * @param name The name of the agent.
     * @param jTextAreaConsole The monitor console.
     */
    public MonitorAgent(String name, JTextArea jTextAreaConsole) {
        this.name = name;
        this.jTextAreaConsole = jTextAreaConsole;
    }

    /**
     * Stops the running agent.
     */
    public void stopAgent() {
        this.running = false;
    }

    /**
     * Override of the run() method. Main implementation of the agent work.
     */
    @Override
    public void run() {
        try {

            // Create TuCSoN Context
            ctx = new TucsonContextSynch(Tucson.getContext(new AgentId(name)));

            // Set Tuple Centre
            TupleCentreId tcid = new TupleCentreId("monitor_centre");
```

```
            // Construct Logic Template
            LogicTuple template = new LogicTuple("monitor", new Var("TupleCentre"),
                 new Var("Tuple"));

            while(running) {

                // Wait for tuple to come
                LogicTuple req = ctx.in(tcid, template);
                appendConsole("----------------------------------------");
                appendConsole((new Date()).toString());
                appendConsole(req.getArg(0).toString() + " : " + req.getArg(1).
                    toString());

            }

            ctx.exit();

        } catch (ContextNotAvailableException ex) {
            appendConsole("ContextNotAvailableException : " + ex.getMessage());
        } catch (InvalidTupleCentreIdException ex) {
            appendConsole("InvalidTupleCentreIdException : " + ex.getMessage());
        } catch (OperationNotAllowedException ex) {
            appendConsole("OperationNotAllowedException : " + ex.getMessage());
        } catch (UnreachableNodeException ex) {
            appendConsole("UnreachableNodeException : " + ex.getMessage());
        } catch (InvalidVarNameException ex) {
            appendConsole("InvalidVarNameException : " + ex.getMessage());
        } catch (InvalidTupleOperationException ex) {
            appendConsole("InvalidTupleOperationException : " + ex.getMessage());
        }

        if(ctx != null) {
            try {
                ctx.exit();
            } catch (OperationNotAllowedException ex) {
                appendConsole("OperationNotAllowedException : " + ex.getMessage());
            }
        }
    }

    /**
     * Appends a new line at the console.
     * @param text The text to add to the console.
     */
    private void appendConsole(String text) {
        if(!jTextAreaConsole.getText().isEmpty())
            jTextAreaConsole.setText(jTextAreaConsole.getText() + "\n" + text);
        else
            jTextAreaConsole.setText(text);
    }

}
```

Listing E.1 – MonitorAgent Class

```
package be.woine.thesis.agent;

import alice.logictuple.*;
import alice.tucson.api.*;
import be.woine.thesis.action.common.ActionType;
import be.woine.thesis.dialog.JDialogAlertProcessing;
import be.woine.thesis.rule.common.CompoundRule;
import be.woine.thesis.rule.common.Rule;
import be.woine.thesis.rule.common.RuleOperator;
import be.woine.thesis.rule.common.RuleType;
import be.woine.thesis.rule.common.ValueSource;
import java.awt.Color;
import java.io.ByteArrayInputStream;
import java.io.IOException;
```

```
import java.io.ObjectInputStream;
import java.util.Date;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.JTextPane;
import javax.swing.text.AttributeSet;
import javax.swing.text.BadLocationException;
import javax.swing.text.Style;
import javax.swing.text.StyleConstants;
import javax.swing.text.StyleContext;
import javax.swing.text.StyledDocument;

/**
 * This class defines the action agent in charge of managing the action and
     triggering the relative method.
 * @author Remy
 * @version 1.0
 */
public class ActionAgent extends Thread {

    private String name;
    private JTextPane textPane;
    private StyledDocument doc;
    private JTextArea jTextAreaConsole;
    private JTextField jTextFieldAlertDialogNumber;
    private Boolean running = true;

    private int numberOfThrownDialog = 0;

    private TucsonContextSynch ctx;

    /**
     * Constructor of the ActionAgent object.
     * @param name The name of the Agent.
     * @param textPane The JTextPane for monitoring the acts of the agent and the
         sequence of actions.
     * @param jTextAreaConsole The console that will be fed in the case of a '
         print_console' action.
     * @param jTextFieldAlertDialogNumber The text field indicating the number of
         dialog displayed in the case of a 'dialog_alert' action.
     */
    public ActionAgent(String name, JTextPane textPane, JTextArea jTextAreaConsole,
         JTextField jTextFieldAlertDialogNumber) {
        this.name = name;
        this.textPane = textPane;
        this.doc = this.textPane.getStyledDocument();
        addStylesToDocument(this.doc);
        this.jTextAreaConsole = jTextAreaConsole;
        this.jTextFieldAlertDialogNumber = jTextFieldAlertDialogNumber;
    }

    /**
     * Adds style to the StyledDocument.
     * @param doc The StyledDocument.
     */
    private void addStylesToDocument(StyledDocument doc) {

        Style def = StyleContext.getDefaultStyleContext().getStyle(StyleContext.
            DEFAULT_STYLE);

        Style regular = doc.addStyle("regular", def);
        StyleConstants.setFontFamily(def, "Tahoma");

        Style s = doc.addStyle("italic", regular);
        StyleConstants.setItalic(s, true);

        s = doc.addStyle("bold", regular);
        StyleConstants.setBold(s, true);

        s = doc.addStyle("red", regular);
```

```
        StyleConstants.setForeground(s, Color.RED);

        s = doc.addStyle("orange", regular);
        StyleConstants.setForeground(s, Color.ORANGE);

        s = doc.addStyle("green", regular);
        StyleConstants.setForeground(s, Color.GREEN);

        s = doc.addStyle("blue", regular);
        StyleConstants.setForeground(s, Color.BLUE);

    }

    /**
     * Appends text to the JTextPane with the correct style.
     * @param line The line to add to the JTextPane.
     * @param a The attribute of the text to add.
     * @param newLine A boolean indicating if the text should be added on a new
          line.
     */
    private void appendTextPane(String line, AttributeSet a, Boolean newLine) {
        try {
            if(newLine)
                doc.insertString(doc.getLength(), line + "\n", a);
            else
                doc.insertString(doc.getLength(), line, a);
        } catch (BadLocationException ex) {
            System.out.println(ex);
        }
    }

    /**
     * Method to stop the running agent.
     */
    public void stopAgent() {
        running = false;
        appendTextPane("Request␣Agent␣stop␣...", doc.getStyle("italic"), true);
    }

    /**
     * Override of the run() methode. Main implementation of the agent work.
     */
    @Override
    public void run() {
        try {

            // Create TuCSoN Context
            appendTextPane("Creating␣new␣TuCsoN␣Context␣with␣Agent␣:␣" + name + "␣
                ...", doc.getStyle("regular"), true);
            ctx = new TucsonContextSynch(Tucson.getContext(new AgentId(name)));
            appendTextPane("TuCSoN␣Context␣created.", doc.getStyle("regular"), true
                );

            // Set Tuple Centre
            appendTextPane("Add␣watch␣to␣Tuple␣Centre␣", doc.getStyle("regular"),
                false);
            appendTextPane("'action_centre'", doc.getStyle("bold"), false);
            appendTextPane("␣...", doc.getStyle("regular"), true);
            TupleCentreId tcid = new TupleCentreId("action_centre");
            TupleCentreId tcid_ex = new TupleCentreId("exchange_centre");

            // Construct Logic Template
            LogicTuple template = new LogicTuple("action", new Var("Type"), new Var
                ("RuleId"));
            LogicTuple template_ex = new LogicTuple("rule", new Var("RuleId"), new
                Var("Object"));

            appendTextPane("Agent␣ready␣!", doc.getStyle("green"), true);

            while(running) {
```

```
            appendTextPane("-------------------------------------------------------"
                , doc.getStyle("regular"), true);
            appendTextPane("Waiting for tuples ...", doc.getStyle("regular"),
                true);

            // Wait for tuple to come
            LogicTuple req = ctx.in(tcid, template);

            appendTextPane("New tuple arrived : ", doc.getStyle("regular"),
                false);
            appendTextPane(req.toString(), doc.getStyle("bold"), true);

            appendTextPane("Processing ...", doc.getStyle("regular"), true);

            // Send request with Rule ID
            ctx.out(tcid_ex, new LogicTuple("rule_req", req.getArg(1)));

            // Wait for response
            LogicTuple rule_req = ctx.in(tcid_ex, template_ex);

            if(!rule_req.getArg(1).isInt()) {

                // Parsing into byte[]
                TupleArgument[] rtup = rule_req.getArg(1).toArray();
                byte[] objectToGet = new byte[rtup.length];

                for(int j = 0; j < rtup.length; j++) {
                    objectToGet[j] = (byte)rtup[j].intValue();
                }

                // Input Stream
                ByteArrayInputStream bais = new ByteArrayInputStream(
                    objectToGet);
                ObjectInputStream ois = new ObjectInputStream(bais);
                ois.close();

                // Read the object
                Rule rule = (Rule)ois.readObject();

                appendTextPane("Rule[" + rule_req.getArg(0).intValue() + "] : "
                    , doc.getStyle("regular"), false);
                appendTextPane(rule.toString(), doc.getStyle("regular"), true);
                System.out.println(rule.toString());

                processAction(rule);

            }

        }

        ctx.exit();

    } catch (ContextNotAvailableException ex) {
        appendTextPane("ContextNotAvailableException : " + ex.getMessage(), doc
            .getStyle("red"), true);
    } catch (InvalidTupleCentreIdException ex) {
        appendTextPane("InvalidTupleCentreIdException : " + ex.getMessage(),
            doc.getStyle("red"), true);
    } catch (OperationNotAllowedException ex) {
        appendTextPane("OperationNotAllowedException : " + ex.getMessage(), doc
            .getStyle("red"), true);
    } catch (UnreachableNodeException ex) {
        appendTextPane("UnreachableNodeException : " + ex.getMessage(), doc.
            getStyle("red"), true);
    } catch (InvalidTupleOperationException ex) {
        appendTextPane("InvalidTupleOperationException : " + ex.getMessage(),
            doc.getStyle("red"), true);
    } catch (InvalidVarNameException ex) {
        appendTextPane("InvalidVarNameException : " + ex.getMessage(), doc.
```

```
                getStyle("red"), true);
        } catch (IOException ex) {
            appendTextPane("IOException : " + ex.getMessage(), doc.getStyle("red"),
                 true);
        } catch (ClassNotFoundException ex) {
            appendTextPane("ClassNotFoundException : " + ex.getMessage(), doc.
                getStyle("red"), true);
        }

        if(ctx != null) {
            try {
                ctx.exit();
            } catch (OperationNotAllowedException ex) {
                appendTextPane("OperationNotAllowedException : " + ex.getMessage(),
                     doc.getStyle("red"), true);
            }
        }

        appendTextPane("Agent stopped.", doc.getStyle("regular"), true);

    }

    /**
     * Processes the triggered action, depending on the action type.
     */
    public void processAction(Rule rule) {
        if(rule.getAction().getType().toString().compareTo(ActionType.PRINT_CONSOLE
            .toString()) == 0) {

            appendConsole("New action in Building : " + rule.getCompoundRule(0).
                getSimpleRule(0).getSource().building.getName());
            appendConsole((new Date()).toString());
            appendConsole("Rule[" + rule.getId() + "] of type : " + rule.getType())
                ;
            appendConsole(getDescriptionFromRule(rule) + "
                ----------------------------------------");

        } else if(rule.getAction().getType().toString().compareTo(ActionType.
            DIALOG_ALERT.toString()) == 0) {

            numberOfThrownDialog++;
            jTextFieldAlertDialogNumber.setText(String.valueOf(numberOfThrownDialog
                ));

            // Display Dialog in non-modal mode
            JDialogAlertProcessing dialog = new JDialogAlertProcessing(null, false)
                ;
            dialog.setLocationByPlatform(true);
            dialog.setRuleId(rule.getId());
            dialog.setRuleType(rule.getType().toString());
            dialog.setBuildingName(rule.getCompoundRule(0).getSimpleRule(0).
                getSource().building.getName());
            dialog.setTime((new Date()).toString());
            dialog.setDescription(getDescriptionFromRule(rule));

            dialog.setVisible(true);

        }

    }

    /**
     * Appends the console in the case of a 'print_console' action.
     * @param text
     */
    private void appendConsole(String text) {
        if(!jTextAreaConsole.getText().isEmpty())
            jTextAreaConsole.setText(jTextAreaConsole.getText() + "\n" + text);
        else
            jTextAreaConsole.setText(text);
```

```
    }

    /**
     * Gets the description of the rule.
     * @param rule The rule from which the description should be obtain.
     * @return A string describing the rule.
     */
    private String getDescriptionFromRule(Rule rule) {

        String desc = "";

        if(rule.getType() == RuleType.FIXED) {
            desc = getDescriptionFromFixedRule(rule);
        } else if(rule.getType() == RuleType.RANGE) {
            desc = getDescriptionFromRangeRule(rule);
        } else if(rule.getType() == RuleType.COMPOUND) {
            desc = getDescriptionFromCompoundRule(rule);
        } else if(rule.getType() == RuleType.SEQUENCE) {
            desc = getDescriptionFromSequenceRule(rule);
        }

        return desc;

    }

    /**
     * Gets the description of the simple fixed rule.
     * @param rule The rule from which the description should be obtain.
     * @return A string describing the simple fixed rule.
     */
    private String getDescriptionFromFixedRule(Rule rule) {

        String desc = "";

        ValueSource source = rule.getCompoundRule(0).getSimpleRule(0).getSource();
        desc += "at floor : '" + source.building.getFloor(source.floorIndex).
            getName();
        desc += "' in room : '" + source.building.getFloor(source.floorIndex).
            getRoom(source.roomIndex).getName();
        desc += "'\n";
        desc += source.building.getFloor(source.floorIndex).getRoom(source.
            roomIndex).getSensor(source.sensorIndex).getType().toString();
        desc += " " + rule.getCompoundRule(0).getSimpleRule(0).getRuleBody(0).
            getOperator().getDescription() + " ";
        desc += rule.getCompoundRule(0).getSimpleRule(0).getRuleBody(0).getValue();
        desc += "\n";

        return desc;

    }

     /**
     * Gets the description of the simple range rule.
     * @param rule The rule from which the description should be obtain.
     * @return A string describing the simple range rule.
     */
    private String getDescriptionFromRangeRule(Rule rule) {

        String desc = "";

        ValueSource source = rule.getCompoundRule(0).getSimpleRule(0).getSource();
        desc += "at floor : '" + source.building.getFloor(source.floorIndex).
            getName();
        desc += "' in room : '" + source.building.getFloor(source.floorIndex).
            getRoom(source.roomIndex).getName();
        desc += "'\n";
        desc += source.building.getFloor(source.floorIndex).getRoom(source.
            roomIndex).getSensor(source.sensorIndex).getType().toString();
        desc += " " + rule.getCompoundRule(0).getSimpleRule(0).getSpecialOperator()
            .getDescription() + " range :\n";
```

```
        desc += "from " + rule.getCompoundRule(0).getSimpleRule(0).getRuleBody(0).
            getValue() + " (";
        if(rule.getCompoundRule(0).getSimpleRule(0).getRuleBody(0).getOperator().
            toString().compareTo(RuleOperator.GREATER_THAN_OR_EQUAL_TO.toString())
            == 0)
            desc += "included";
        else
            desc += "excluded";
        desc += ") to " + rule.getCompoundRule(0).getSimpleRule(0).getRuleBody(1).
            getValue() + " (";
        if(rule.getCompoundRule(0).getSimpleRule(0).getRuleBody(1).getOperator().
            toString().compareTo(RuleOperator.SMALLER_THAN_OR_EQUAL_TO.toString())
            == 0)
            desc += "included";
        else
            desc += "excluded";
        desc += ")";
        desc += "\n";

        return desc;

    }

    /**
     * Gets the description of the compound rule.
     * @param rule The rule from which the description should be obtain.
     * @return A string describing the compound rule.
     */
    private String getDescriptionFromCompoundRule(Rule rule) {

        String desc = "";

        desc += "Compound rule composed of " + rule.getCompoundRule(0).
            getSimpleRuleList().size() + " terms\n";

        for(int i = 0; i < rule.getCompoundRule(0).getSimpleRuleList().size(); i++)
             {

            desc += "- Term " + i + " :\n";

            Rule innerRule = new Rule();
            innerRule.addCompoundRule(new CompoundRule());
            innerRule.getCompoundRule(0).addSimpleRule(rule.getCompoundRule(0).
                getSimpleRule(i));

            if(rule.getCompoundRule(0).getSimpleRule(i).getType() == RuleType.FIXED
                )
                desc += getDescriptionFromFixedRule(innerRule);
            else if(rule.getCompoundRule(0).getSimpleRule(i).getType() == RuleType.
                RANGE)
                desc += getDescriptionFromRangeRule(innerRule);

        }

        return desc;

    }

    /**
     * Gets the description of the sequence rule.
     * @param rule The rule from which the description should be obtain.
     * @return A string describing the sequence rule.
     */
    private String getDescriptionFromSequenceRule(Rule rule) {

        String desc = "";

        desc += "Sequence rule composed of " + rule.getCompoundRuleList().size() +
            " steps\n";
```

```
        for(int i = 0; i < rule.getCompoundRuleList().size(); i++) {

            desc += "* Step " + i + " :\n";

            Rule innerRule = new Rule();
            innerRule.addCompoundRule(rule.getCompoundRule(i));

            if(rule.getCompoundRule(i).getType() == RuleType.FIXED)
                desc += getDescriptionFromFixedRule(innerRule);
            else if(rule.getCompoundRule(i).getType() == RuleType.RANGE)
                desc += getDescriptionFromRangeRule(innerRule);
            else if(rule.getCompoundRule(i).getType() == RuleType.COMPOUND)
                desc += getDescriptionFromCompoundRule(innerRule);

        }

        return desc;

    }

}
```

Listing E.2 – ActionAgent Class

```
package be.woine.thesis.agent;

import alice.logictuple.LogicTuple;
import alice.logictuple.TupleArgument;
import alice.logictuple.Value;
import alice.tucson.api.AgentId;
import alice.tucson.api.ContextNotAvailableException;
import alice.tucson.api.InvalidTupleCentreIdException;
import alice.tucson.api.OperationNotAllowedException;
import alice.tucson.api.Tucson;
import alice.tucson.api.TucsonContextSynch;
import alice.tucson.api.TupleCentreId;
import alice.tucson.api.UnreachableNodeException;
import be.woine.thesis.building.common.Building;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

/**
 *
 * @author Remy
 * @version 1.0
 */
public class BuildingAgent extends Thread {

    private String name;
    private Building building;
    private TucsonContextSynch ctx;

    public BuildingAgent(String name, Building building) {
        this.name = name;
        this.building = building;
    }

    @Override
    public void run() {
        try {

            ctx = new TucsonContextSynch(Tucson.getContext(new AgentId(name)));
            TupleCentreId tcid = new TupleCentreId("exchange_centre");

            LogicTuple template = new LogicTuple("building_req");

            while(true) {
```

```
                // Wait for request
                LogicTuple req = ctx.in(tcid, template);

                // Output Stream
                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                ObjectOutputStream oos = new ObjectOutputStream(baos);

                // Write the object
                oos.writeObject(building);
                oos.close();

                // Parsing into TupleArgument
                byte[] objectToSend = baos.toByteArray();
                TupleArgument[] tup = new TupleArgument[objectToSend.length];

                for(int i = 0; i < objectToSend.length; i++)
                    tup[i] = new Value(objectToSend[i]);

                // Send in tuple centre
                LogicTuple t = new LogicTuple("building", new Value(tup));
                ctx.out(tcid, t);

            }

            //ctx.exit();

        } catch (ContextNotAvailableException ex) {
            System.err.println(ex);
        } catch (InvalidTupleCentreIdException ex) {
            System.err.println(ex);
        } catch (OperationNotAllowedException ex) {
            System.err.println(ex);
        } catch (UnreachableNodeException ex) {
            System.err.println(ex);
        } catch (IOException ex) {
            System.err.println(ex);
        }

        if(ctx != null) {
            try {
                ctx.exit();
            } catch (OperationNotAllowedException ex) {
                System.err.println(ex);
            }
        }
    }

}
```

Listing E.3 – BuildingAgent Class

```
package be.woine.thesis.agent;

import alice.logictuple.InvalidTupleOperationException;
import alice.logictuple.InvalidVarNameException;
import alice.logictuple.LogicTuple;
import alice.logictuple.TupleArgument;
import alice.logictuple.Value;
import alice.logictuple.Var;
import alice.tucson.api.AgentId;
import alice.tucson.api.ContextNotAvailableException;
import alice.tucson.api.InvalidTupleCentreIdException;
import alice.tucson.api.OperationNotAllowedException;
import alice.tucson.api.Tucson;
import alice.tucson.api.TucsonContextSynch;
import alice.tucson.api.TupleCentreId;
import alice.tucson.api.UnreachableNodeException;
import be.woine.thesis.rule.common.Rule;
import java.io.ByteArrayOutputStream;
```

```
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.ArrayList;

/**
 *
 * @author Remy
 * @version 1.0
 */
public class RulerAgent extends Thread {

    private String name;
    private TucsonContextSynch ctx;
    private ArrayList<Rule> ruleList;
    private ArrayList<Rule> newRuleList;

    public RulerAgent(String name, ArrayList<Rule> ruleList) {
        this.name = name;
        this.ruleList = ruleList;
        this.newRuleList = ruleList;
    }

    public void updateRuleList(ArrayList<Rule> ruleList) {
        this.newRuleList = ruleList;
    }

    @Override
    public void run() {
        try {

            ctx = new TucsonContextSynch(Tucson.getContext(new AgentId(name)));
            TupleCentreId tcid = new TupleCentreId("exchange_centre");

            LogicTuple template = new LogicTuple("rule_req", new Var("RuleId"));

            while(true) {

                // Wait for request
                LogicTuple req = ctx.in(tcid, template);

                // Get Rule from ID
                Rule rule = getRuleFromId(req.getArg(0).intValue());

                // Output Stream
                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                ObjectOutputStream oos = new ObjectOutputStream(baos);

                // Write the object
                oos.writeObject(rule);
                oos.close();

                // Parsing into TupleArgument
                byte[] objectToSend = baos.toByteArray();
                TupleArgument[] tup = new TupleArgument[objectToSend.length];

                for(int i = 0; i < objectToSend.length; i++)
                    tup[i] = new Value(objectToSend[i]);

                // Send in tuple centre
                LogicTuple t = new LogicTuple("rule", new Value(req.getArg(0).
                    intValue()), new Value(tup));
                ctx.out(tcid, t);

            }

            //ctx.exit();

        } catch (ContextNotAvailableException ex) {
            System.err.println(ex);
        } catch (InvalidTupleCentreIdException ex) {
```

```
            System.err.println(ex);
        } catch (OperationNotAllowedException ex) {
            System.err.println(ex);
        } catch (UnreachableNodeException ex) {
            System.err.println(ex);
        } catch (InvalidTupleOperationException ex) {
            System.err.println(ex);
        } catch (InvalidVarNameException ex) {
            System.err.println(ex);
        } catch (IOException ex) {
            System.err.println(ex);
        }

        if(ctx != null) {
            try {
                ctx.exit();
            } catch (OperationNotAllowedException ex) {
                System.err.println(ex);
            }
        }
    }

    public Rule getRuleFromId(int id) {

        Rule rule = null;

        // Update
        this.ruleList = this.newRuleList;

        for(int i = 0; i < this.ruleList.size(); i++) {
            if(this.ruleList.get(i).getId() == id) {
                rule = this.ruleList.get(i);
                break;
            }
        }

        return rule;

    }

}
```

Listing E.4 – RulerAgent Class

Annexe F

package be.woine.thesis.node.common

```java
package be.woine.thesis.node.common;

import java.util.ArrayList;

/**
 * This class defines the environment's node used in the application.
 * @author Remy
 * @version 1.0
 */
public class NodeEnvironment {

    private String name;
    private ArrayList<TupleCentre> tupleCentreList;

    /**
     * Empty constructor for the NodeEnvironment object.
     */
    public NodeEnvironment() {
        this.name = "";
        this.tupleCentreList = new ArrayList<TupleCentre>();
    }

    /**
     * Constructor of the NodeEnvironment object.
     * @param name The name of the node.
     */
    public NodeEnvironment(String name) {
        this.name = name;
        this.tupleCentreList = new ArrayList<TupleCentre>();
    }

    /**
     * Constructor of the NodeEnvironment object.
     * @param name The name of the node.
     * @param tupleCentreList The list of tuple centre contained in the node.
     */
    public NodeEnvironment(String name, ArrayList<TupleCentre> tupleCentreList) {
        this.name = name;
        this.tupleCentreList = tupleCentreList;
    }

    /**
     * Gets the name of the node.
     * @return The name of the node.
     */
    public String getName() {
        return name;
    }
```

```
    /**
     * Sets the name of the node.
     * @param name The new name of the node.
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Gets the list of the tuple centre contained in the node.
     * @return The list of the tuple centre contained in the node.
     */
    public ArrayList<TupleCentre> getTupleCentreList() {
        return tupleCentreList;
    }

    /**
     * Sets the list of the tuple centre in the node.
     * @param tupleCentreList The list of the tuple centre in the node.
     */
    public void setTupleCentreList(ArrayList<TupleCentre> tupleCentreList) {
        this.tupleCentreList = tupleCentreList;
    }

    /**
     * Adds a tuple centre in the node.
     * @param tupleCentre The new tuple centre to add in the node.
     */
    public void addTupleCentre(TupleCentre tupleCentre) {
        this.tupleCentreList.add(tupleCentre);
    }

    /**
     * Gets a tuple centre from the node at a specific index.
     * @param index The index of the tuple centre to get.
     * @return The tuple centre at the mentionned index.
     */
    public TupleCentre getTupleCentre(int index) {
        return this.tupleCentreList.get(index);
    }

    /**
     * Gets the number of tuple centre in the node.
     * @return The number of tuple centre in the node.
     */
    public int getTupleCentreCount() {
        return this.tupleCentreList.size();
    }

    /**
     * Override of the toString() method to get a correct string representing the
       node.
     * @return The name of the node.
     */
    @Override
    public String toString() {
        return this.name;
    }

}
```

Listing F.1 – NodeEnvironment Class

```
package be.woine.thesis.node.common;

import be.woine.thesis.reaction.common.Reaction;
import java.util.ArrayList;
```

```java
/**
 * This class defines the a tuple centre object as it will be used in the
     application.
 * This class is not equivalent to the alice.respect.TupleCentre class.
 * @author Remy
 * @version 1.0
 */
public class TupleCentre implements Comparable {

    private String id;
    private ArrayList<Reaction> reactionList;

    /**
     * Empty constructor of the TupleCentre object.
     */
    public TupleCentre() {
        this.id = "";
        this.reactionList = new ArrayList<Reaction>();
    }

    /**
     * Constructor of the TupleCentre object.
     * @param id Unique ID of the tuple centre.
     */
    public TupleCentre(String id) {
        this.id = id;
        this.reactionList = new ArrayList<Reaction>();
    }

    /**
     * Constructor of the TupleCentre object.
     * @param id Unique ID of the tuple centre.
     * @param reactionList List of reactions contained in the tuple centre.
     */
    public TupleCentre(String id, ArrayList<Reaction> reactionList) {
        this.id = id;
        this.reactionList = reactionList;
    }

    /**
     * Gets the tuple centre id.
     * @return The tuple centre unique id.
     */
    public String getId() {
        return id;
    }

    /**
     * Sets the tuple centre id.
     * @param id The new tuple centre id.
     */
    public void setId(String id) {
        this.id = id;
    }

    /**
     * Gets the list of reactions contained in the tuple centre.
     * @return The list of reactions contained in the tuple centre.
     */
    public ArrayList<Reaction> getReactionList() {
        return reactionList;
    }

    /**
     * Sets the list of reaction to the tuple centre.
     * @param reactionList The reaction list to set to the tuple centre.
     */
    public void setReactionList(ArrayList<Reaction> reactionList) {
        this.reactionList = reactionList;
    }
```

```
    /**
     * Adds a reaction to the tuple centre.
     * @param reaction The reaction to add to the tuple centre.
     */
    public void addReaction(Reaction reaction) {
        this.reactionList.add(reaction);
    }

    /**
     * Gets the reaction from the tuple centre at the mentionned index.
     * @param index Index of the reaction to get.
     * @return The reaction at the mentionned index.
     */
    public Reaction getReaction(int index) {
        return this.reactionList.get(index);
    }

    /**
     * Gets the number of reaction in the tuple centre.
     * @return The number of reaction in the tuple centre.
     */
    public int getReactionCount() {
        return this.reactionList.size();
    }

    /**
     * Overrie of the toString() method to get a correct string representing the
         TupleCentre.
     * @return The unique ID of the tuple centre.
     */
    @Override
    public String toString() {
        return this.id;
    }

    /**
     * Implements the compareTo() method to easy the sorting of a list of tuple
         centre.
     * @param o The other TupleCentre to compare with this one.
     * @return An int value representing the comparison between two TupleCentre
         objects.
     */
    @Override
    public int compareTo(Object o) {

        int value = this.getId().compareToIgnoreCase(((TupleCentre)o).getId());

        if(value > 1)
            value = 1;

        return value;

    }

}
```

Listing F.2 – TupleCentre Class

Annexe G

package be.woine.thesis.reaction.common

```java
package be.woine.thesis.reaction.common;

/**
 * This class defines the Reaction object used for the tuple centre specification.
 * @author Remy
 * @version 1.0
 */
public class Reaction {

    private String tupleCentreId;
    private String event;
    private String body;
    private Boolean settled = false;

    /**
     * Empty constructor for the Reaction object.
     */
    public Reaction() {
        this.event = "";
        this.body = "";
        tupleCentreId = "";
    }

    /**
     * Constructor of the Reaction object.
     * @param event The event triggering the reaction.
     * @param body The body of the reaction containing all the predicates.
     */
    public Reaction(String event, String body) {
        this.event = event;
        this.body = body;
    }

    /**
     * Constructor of the Reaction object.
     * @param event The event triggering the reaction.
     * @param body The body of the reaction containing all the predicates.
     * @param tupleCentreId The tuple centre ID in which the reaction should be
         programmed.
     */
    public Reaction(String event, String body, String tupleCentreId) {
        this.event = event;
        this.body = body;
        this.tupleCentreId = tupleCentreId;
    }

    /**
     * Gets the body of the reaction.
```

```
     * @return The body of the reaction as a string.
     */
    public String getBody() {
        return body;
    }

    /**
     * Sets the body of the reaction.
     * @param body The body of the reaction to set.
     */
    public void setBody(String body) {
        this.body = body;
    }

    /**
     * Appends a predicate to the body of the reaction.
     * @param predicate The predicate to append to the body.
     */
    public void appendBody(String predicate) {
        if(this.body.length() == 0) {
            this.body = predicate;
        } else {
            this.body += "," + predicate;
        }
    }

    /**
     * Gets the event triggering the reaction.
     * @return The event of the reaction.
     */
    public String getEvent() {
        return event;
    }

    /**
     * Sets the event triggering the reaction.
     * @param event The event of the reaction.
     */
    public void setEvent(String event) {
        this.event = event;
    }

    /**
     * Gets the tuple centre ID in which the reaction is programmed.
     * @return The ID of the tuple centre in which the reaction is programmed.
     */
    public String getTupleCentreId() {
        return tupleCentreId;
    }

    /**
     * Sets the tuple centre ID in which the reacion should be programmed.
     * @param tupleCentreId The ID of the tuple centre to set.
     */
    public void setTupleCentreId(String tupleCentreId) {
        this.tupleCentreId = tupleCentreId;
    }

    /**
     * Tells if the reaction has been settled in the tuple centre.
     * @return True is the reaction has been settled, false otherwise.
     */
    public Boolean isSettled() {
        return this.settled;
    }

    /**
     * Configures if the reaction has been settled or not in the tuple centre.
     * @param status Set to true if the reaction is settled in the tuple centre,
         false otherwise.
```

```
     */
    public void setSettledStatus(Boolean status) {
        this.settled = status;
    }

    /**
     * Override of the toString() method to get a correct string representing the
         reaction.
     * @return A string representing the reaction. This string can be programmed in
         the tuple centre as-is.
     */
    @Override
    public String toString() {
        return "reaction(" + event + ", (" + body + ")).\n";
    }

}
```

Listing G.1 – Reaction Class

Annexe H

package be.woine.thesis.dialog

```
/*
 * JDialogCompoundRule.java
 *
 * Created on 18-mai-2012, 12:56:17
 */
package be.woine.thesis.dialog;

import be.woine.thesis.building.common.Building;
import be.woine.thesis.model.CompoundRuleAbstractTableModel;
import be.woine.thesis.rule.common.CompoundRule;
import be.woine.thesis.rule.common.RuleType;
import javax.swing.JOptionPane;

/**
 *
 * @author Remy
 */
public class JDialogCompoundRule extends javax.swing.JDialog {

    private Building building;
    private CompoundRule compoundRule = new CompoundRule();

    private int returnValue = JOptionPane.CANCEL_OPTION;

    /** Creates new form JDialogCompoundRule */
    public JDialogCompoundRule(Building building, java.awt.Frame parent, boolean
        modal) {
        super(parent, modal);

        this.building = building;

        /* Look And Feel */
        try {
            javax.swing.UIManager.setLookAndFeel(javax.swing.UIManager.
                getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException ex) {
            System.out.println(ex.toString());
        } catch (InstantiationException ex) {
            System.out.println(ex.toString());
        } catch (IllegalAccessException ex) {
            System.out.println(ex.toString());
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            System.out.println(ex.toString());
        }

        initComponents();
    }

    public int getReturnValue() {
        return returnValue;
    }
```

```
public CompoundRule getCompoundRule() {
    return compoundRule;
}


/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:
    initComponents
private void initComponents() {

    jPanelCompoundRule = new javax.swing.JPanel();
    jScrollPaneRuleTable = new javax.swing.JScrollPane();
    jTableRule = new javax.swing.JTable();
    jButtonAdd = new javax.swing.JButton();
    jButtonEdit = new javax.swing.JButton();
    jButtonRemove = new javax.swing.JButton();
    jButtonCancel = new javax.swing.JButton();
    jButtonOk = new javax.swing.JButton();
    jLabelNumberOfTerms = new javax.swing.JLabel();

    setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
    setTitle("Compound Rule");

    jPanelCompoundRule.setBorder(javax.swing.BorderFactory.createTitledBorder(
        null, "Compound Rule", javax.swing.border.TitledBorder.
        DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.DEFAULT_POSITION
        , new java.awt.Font("Tahoma", 1, 11), new java.awt.Color(0, 0, 255)));
        // NOI18N

    jTableRule.setModel(new CompoundRuleAbstractTableModel());
    jScrollPaneRuleTable.setViewportView(jTableRule);

    jButtonAdd.setText("Add");
    jButtonAdd.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonAddActionPerformed(evt);
        }
    });

    jButtonEdit.setText("Edit");
    jButtonEdit.setEnabled(false);

    jButtonRemove.setText("Remove");
    jButtonRemove.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonRemoveActionPerformed(evt);
        }
    });

    jButtonCancel.setText("Cancel");
    jButtonCancel.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonCancelActionPerformed(evt);
        }
    });

    jButtonOk.setText("OK");
    jButtonOk.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonOkActionPerformed(evt);
        }
    });

    jLabelNumberOfTerms.setText("Number of terms = 0");
```

```
javax.swing.GroupLayout jPanelCompoundRuleLayout = new javax.swing.
    GroupLayout(jPanelCompoundRule);
jPanelCompoundRule.setLayout(jPanelCompoundRuleLayout);
jPanelCompoundRuleLayout.setHorizontalGroup(
    jPanelCompoundRuleLayout.createParallelGroup(javax.swing.GroupLayout.
        Alignment.LEADING)
    .addGroup(jPanelCompoundRuleLayout.createSequentialGroup()
        .addContainerGap()
        .addGroup(jPanelCompoundRuleLayout.createParallelGroup(javax.swing.
            GroupLayout.Alignment.LEADING, false)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                jPanelCompoundRuleLayout.createSequentialGroup()
                .addComponent(jLabelNumberOfTerms)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                    .RELATED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
                .addComponent(jButtonOk, javax.swing.GroupLayout.
                    PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                    PREFERRED_SIZE))
            .addComponent(jScrollPaneRuleTable, javax.swing.GroupLayout.
                PREFERRED_SIZE, 223, javax.swing.GroupLayout.PREFERRED_SIZE
                ))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
            UNRELATED)
        .addGroup(jPanelCompoundRuleLayout.createParallelGroup(javax.swing.
            GroupLayout.Alignment.TRAILING)
            .addGroup(jPanelCompoundRuleLayout.createParallelGroup(javax.
                swing.GroupLayout.Alignment.LEADING)
                .addComponent(jButtonAdd, javax.swing.GroupLayout.
                    PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                    PREFERRED_SIZE)
                .addComponent(jButtonEdit, javax.swing.GroupLayout.
                    PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                    PREFERRED_SIZE)
                .addComponent(jButtonRemove, javax.swing.GroupLayout.
                    PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                    PREFERRED_SIZE))
            .addComponent(jButtonCancel, javax.swing.GroupLayout.
                PREFERRED_SIZE, 73, javax.swing.GroupLayout.PREFERRED_SIZE)
                )
        .addContainerGap())
);
jPanelCompoundRuleLayout.setVerticalGroup(
    jPanelCompoundRuleLayout.createParallelGroup(javax.swing.GroupLayout.
        Alignment.LEADING)
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
        jPanelCompoundRuleLayout.createSequentialGroup()
        .addContainerGap()
        .addGroup(jPanelCompoundRuleLayout.createParallelGroup(javax.swing.
            GroupLayout.Alignment.TRAILING)
            .addGroup(jPanelCompoundRuleLayout.createSequentialGroup()
                .addComponent(jButtonAdd)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                    .RELATED)
                .addComponent(jButtonEdit)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                    .RELATED)
                .addComponent(jButtonRemove)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                    .RELATED, 125, Short.MAX_VALUE)
                .addComponent(jButtonCancel))
            .addGroup(jPanelCompoundRuleLayout.createParallelGroup(javax.
                swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jButtonOk)
                .addComponent(jLabelNumberOfTerms))
            .addComponent(jScrollPaneRuleTable, javax.swing.GroupLayout.
                Alignment.LEADING, javax.swing.GroupLayout.PREFERRED_SIZE,
                192, javax.swing.GroupLayout.PREFERRED_SIZE))
        .addContainerGap())
```

```
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane
            ());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelCompoundRule, javax.swing.GroupLayout.
                    PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.
                    swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelCompoundRule, javax.swing.GroupLayout.
                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jButtonOkActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
        FIRST:event_jButtonOkActionPerformed

        if(jTableRule.getRowCount() > 0) {

            compoundRule.setSimpleRuleList(((CompoundRuleAbstractTableModel)
                jTableRule.getModel()).getRuleList());

            returnValue = JOptionPane.OK_OPTION;
            setVisible(false);
        } else {
            JOptionPane.showMessageDialog(this,
                "No rule specified.",
                "Error",
                JOptionPane.ERROR_MESSAGE);
        }

    }//GEN-LAST:event_jButtonOkActionPerformed

    private void jButtonCancelActionPerformed(java.awt.event.ActionEvent evt) {//
        GEN-FIRST:event_jButtonCancelActionPerformed
        returnValue = JOptionPane.CANCEL_OPTION;
        setVisible(false);
    }//GEN-LAST:event_jButtonCancelActionPerformed

    private void jButtonAddActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
        FIRST:event_jButtonAddActionPerformed

        JDialogRuleChooser ruleDialog = new JDialogRuleChooser(null, true);
        ruleDialog.setLocationByPlatform(true);
        ruleDialog.jRadioButtonCompound.setEnabled(false);
        ruleDialog.jRadioButtonSequence.setEnabled(false);
        ruleDialog.jComboBoxActionType.setEnabled(false);
        ruleDialog.setVisible(true);

        if(ruleDialog.getReturnValue() == JOptionPane.OK_OPTION) {
            if(ruleDialog.getRuleType() == RuleType.FIXED) {

                JDialogSimpleRuleFixed fixedDialog = new JDialogSimpleRuleFixed(
                    building, null, true);
                fixedDialog.setLocationByPlatform(true);
                fixedDialog.setVisible(true);
```

```
                if(fixedDialog.getReturnValue() == JOptionPane.OK_OPTION) {
                    ((CompoundRuleAbstractTableModel)jTableRule.getModel()).addRule
                        (fixedDialog.getSimpleRule());
                }

            } else if(ruleDialog.getRuleType() == RuleType.RANGE) {

                JDialogSimpleRuleRange rangeDialog = new JDialogSimpleRuleRange(
                    building, null, true);
                rangeDialog.setLocationByPlatform(true);
                rangeDialog.setVisible(true);

                if(rangeDialog.getReturnValue() == JOptionPane.OK_OPTION) {
                    ((CompoundRuleAbstractTableModel)jTableRule.getModel()).addRule
                        (rangeDialog.getSimpleRule());
                }

            }
        }

        jLabelNumberOfTerms.setText("Number of terms = " + jTableRule.getRowCount()
            );

    }//GEN-LAST:event_jButtonAddActionPerformed

    private void jButtonRemoveActionPerformed(java.awt.event.ActionEvent evt) {//
        GEN-FIRST:event_jButtonRemoveActionPerformed
        if(jTableRule.getSelectedRow() >= 0)
            ((CompoundRuleAbstractTableModel)jTableRule.getModel()).removeRule(
                jTableRule.getSelectedRow());

        jLabelNumberOfTerms.setText("Number of terms = " + jTableRule.getRowCount()
            );
    }//GEN-LAST:event_jButtonRemoveActionPerformed

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButtonAdd;
    private javax.swing.JButton jButtonCancel;
    private javax.swing.JButton jButtonEdit;
    private javax.swing.JButton jButtonOk;
    private javax.swing.JButton jButtonRemove;
    private javax.swing.JLabel jLabelNumberOfTerms;
    private javax.swing.JPanel jPanelCompoundRule;
    private javax.swing.JScrollPane jScrollPaneRuleTable;
    private javax.swing.JTable jTableRule;
    // End of variables declaration//GEN-END:variables
}
```

Listing H.1 – JDialogCompoundRule Class

```
/*
 * JDialogRuleChooser.java
 *
 * Created on 18-mai-2012, 8:57:21
 */
package be.woine.thesis.dialog;

import be.woine.thesis.action.common.ActionType;
import be.woine.thesis.rule.common.RuleType;
import javax.swing.JOptionPane;

/**
 *
 * @author Remy
 */
public class JDialogRuleChooser extends javax.swing.JDialog {

    private RuleType ruleType = RuleType.UNDEFINED;
```

```
    private ActionType actionType = ActionType.UNDEFINED;
    private int returnValue = JOptionPane.CANCEL_OPTION;

    /** Creates new form JDialogRuleChooser */
    public JDialogRuleChooser(java.awt.Frame parent, boolean modal) {
        super(parent, modal);

        /* Look And Feel */
        try {
            javax.swing.UIManager.setLookAndFeel(javax.swing.UIManager.
                getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException ex) {
            System.out.println(ex.toString());
        } catch (InstantiationException ex) {
            System.out.println(ex.toString());
        } catch (IllegalAccessException ex) {
            System.out.println(ex.toString());
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            System.out.println(ex.toString());
        }

        initComponents();

        jComboBoxActionType.removeAllItems();
        jComboBoxActionType.addItem(ActionType.PRINT_CONSOLE);
        jComboBoxActionType.addItem(ActionType.DIALOG_ALERT);

    }

    public int getReturnValue() {
        return returnValue;
    }

    public RuleType getRuleType() {
        return ruleType;
    }

    public ActionType getActionType() {
        return actionType;
    }


    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:
        initComponents
    private void initComponents() {

        buttonGroupRule = new javax.swing.ButtonGroup();
        jPanelRuleChooser = new javax.swing.JPanel();
        jRadioButtonFixed = new javax.swing.JRadioButton();
        jRadioButtonRange = new javax.swing.JRadioButton();
        jRadioButtonCompound = new javax.swing.JRadioButton();
        jRadioButtonSequence = new javax.swing.JRadioButton();
        jButtonCancel = new javax.swing.JButton();
        jButtonOk = new javax.swing.JButton();
        jLabelActionToTrigger = new javax.swing.JLabel();
        jComboBoxActionType = new javax.swing.JComboBox();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Rule Chooser");

        jPanelRuleChooser.setBorder(javax.swing.BorderFactory.createTitledBorder(
            null, "Rule Chooser", javax.swing.border.TitledBorder.
            DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.DEFAULT_POSITION
            , new java.awt.Font("Tahoma", 1, 11), new java.awt.Color(0, 0, 255)));
```

```
        // NOI18N

    buttonGroupRule.add(jRadioButtonFixed);
    jRadioButtonFixed.setSelected(true);
    jRadioButtonFixed.setText("Simple Rule (Fixed Value)");

    buttonGroupRule.add(jRadioButtonRange);
    jRadioButtonRange.setText("Simple Rule (Range Value)");

    buttonGroupRule.add(jRadioButtonCompound);
    jRadioButtonCompound.setText("Compound Rule");

    buttonGroupRule.add(jRadioButtonSequence);
    jRadioButtonSequence.setText("Sequence Rule");

    jButtonCancel.setText("Cancel");
    jButtonCancel.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonCancelActionPerformed(evt);
        }
    });

    jButtonOk.setText("OK");
    jButtonOk.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonOkActionPerformed(evt);
        }
    });

    jLabelActionToTrigger.setText("Action to trigger :");

    jComboBoxActionType.setModel(new javax.swing.DefaultComboBoxModel(new
        String[] { "Item 1", "Item 2", "Item 3", "Item 4" }));

    javax.swing.GroupLayout jPanelRuleChooserLayout = new javax.swing.
        GroupLayout(jPanelRuleChooser);
    jPanelRuleChooser.setLayout(jPanelRuleChooserLayout);
    jPanelRuleChooserLayout.setHorizontalGroup(
        jPanelRuleChooserLayout.createParallelGroup(javax.swing.GroupLayout.
            Alignment.LEADING)
        .addGroup(jPanelRuleChooserLayout.createSequentialGroup()
            .addContainerGap()
            .addGroup(jPanelRuleChooserLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.LEADING)
                .addComponent(jRadioButtonFixed)
                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                    jPanelRuleChooserLayout.createSequentialGroup()
                    .addComponent(jButtonOk, javax.swing.GroupLayout.
                        PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                        PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                        .RELATED)
                    .addComponent(jButtonCancel, javax.swing.GroupLayout.
                        PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                        PREFERRED_SIZE))
                .addComponent(jRadioButtonRange)
                .addComponent(jRadioButtonCompound)
                .addComponent(jRadioButtonSequence)
                .addGroup(jPanelRuleChooserLayout.createSequentialGroup()
                    .addComponent(jLabelActionToTrigger)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                        .RELATED)
                    .addComponent(jComboBoxActionType, 0, 120, Short.MAX_VALUE)
                        ))
            .addContainerGap())
    );
    jPanelRuleChooserLayout.setVerticalGroup(
        jPanelRuleChooserLayout.createParallelGroup(javax.swing.GroupLayout.
            Alignment.LEADING)
        .addGroup(jPanelRuleChooserLayout.createSequentialGroup()
```

```
                .addContainerGap()
                .addComponent(jRadioButtonFixed)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
                    UNRELATED)
                .addComponent(jRadioButtonRange)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
                    )
                .addComponent(jRadioButtonCompound)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
                    )
                .addComponent(jRadioButtonSequence)
                .addGap(18, 18, 18)
                .addGroup(jPanelRuleChooserLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabelActionToTrigger)
                    .addComponent(jComboBoxActionType, javax.swing.GroupLayout.
                        PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                        .swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
                    , 25, Short.MAX_VALUE)
                .addGroup(jPanelRuleChooserLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.BASELINE)
                    .addComponent(jButtonCancel)
                    .addComponent(jButtonOk))
                .addContainerGap())
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane
            ());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelRuleChooser, javax.swing.GroupLayout.
                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelRuleChooser, javax.swing.GroupLayout.
                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jButtonOkActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
        FIRST:event_jButtonOkActionPerformed

        if(jRadioButtonFixed.isSelected())
            ruleType = RuleType.FIXED;
        else if(jRadioButtonRange.isSelected())
            ruleType = RuleType.RANGE;
        else if(jRadioButtonCompound.isSelected())
            ruleType = RuleType.COMPOUND;
        else
            ruleType = RuleType.SEQUENCE;

        actionType = (ActionType)jComboBoxActionType.getSelectedItem();

        returnValue = JOptionPane.OK_OPTION;
        this.setVisible(false);
    }//GEN-LAST:event_jButtonOkActionPerformed
```

```
    private void jButtonCancelActionPerformed(java.awt.event.ActionEvent evt) {//
        GEN-FIRST:event_jButtonCancelActionPerformed
        returnValue = JOptionPane.CANCEL_OPTION;
        this.setVisible(false);
    }//GEN-LAST:event_jButtonCancelActionPerformed

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.ButtonGroup buttonGroupRule;
    private javax.swing.JButton jButtonCancel;
    private javax.swing.JButton jButtonOk;
    public javax.swing.JComboBox jComboBoxActionType;
    private javax.swing.JLabel jLabelActionToTrigger;
    private javax.swing.JPanel jPanelRuleChooser;
    public javax.swing.JRadioButton jRadioButtonCompound;
    public javax.swing.JRadioButton jRadioButtonFixed;
    public javax.swing.JRadioButton jRadioButtonRange;
    public javax.swing.JRadioButton jRadioButtonSequence;
    // End of variables declaration//GEN-END:variables
}
```

Listing H.2 – JDialogRuleChooser Class

```
/*
 * JDialogSensor.java
 *
 * Created on 18-mai-2012, 17:42:01
 */
package be.woine.thesis.dialog;

import be.woine.thesis.building.common.SensorType;
import javax.swing.JOptionPane;

/**
 *
 * @author Remy
 */
public class JDialogSensor extends javax.swing.JDialog {

    private int returnValue = JOptionPane.CANCEL_OPTION;
    private String sensorId = "";
    private SensorType sensorType = SensorType.UNDEFINED;

    /** Creates new form JDialogSensor */
    public JDialogSensor(java.awt.Frame parent, boolean modal) {
        super(parent, modal);

        /* Look And Feel */
        try {
            javax.swing.UIManager.setLookAndFeel(javax.swing.UIManager.
                getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException ex) {
            System.out.println(ex.toString());
        } catch (InstantiationException ex) {
            System.out.println(ex.toString());
        } catch (IllegalAccessException ex) {
            System.out.println(ex.toString());
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            System.out.println(ex.toString());
        }

        initComponents();

        jComboBoxType.removeAllItems();
        jComboBoxType.addItem(SensorType.TEMPERATURE);
        jComboBoxType.addItem(SensorType.HUMIDITY);
        jComboBoxType.addItem(SensorType.LUMINOSITY);
        jComboBoxType.addItem(SensorType.PRESSURE);

    }
```

```
public int getReturnValue() {
    return returnValue;
}

public String getSensorId() {
    return sensorId;
}

public SensorType getSensorType() {
    return sensorType;
}


/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:
    initComponents
private void initComponents() {

    jPanelSensor = new javax.swing.JPanel();
    jLabelId = new javax.swing.JLabel();
    jTextFieldId = new javax.swing.JTextField();
    jLabelType = new javax.swing.JLabel();
    jComboBoxType = new javax.swing.JComboBox();
    jButtonCancel = new javax.swing.JButton();
    jButtonOk = new javax.swing.JButton();

    setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);

    jPanelSensor.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "
        Sensor", javax.swing.border.TitledBorder.DEFAULT_JUSTIFICATION, javax.
        swing.border.TitledBorder.DEFAULT_POSITION, new java.awt.Font("Tahoma",
         1, 11), new java.awt.Color(0, 0, 255))); // NOI18N

    jLabelId.setText("ID : ");

    jLabelType.setText("Type : ");

    jComboBoxType.setModel(new javax.swing.DefaultComboBoxModel(new String[] {
        "Item 1", "Item 2", "Item 3", "Item 4" }));

    jButtonCancel.setText("Cancel");
    jButtonCancel.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonCancelActionPerformed(evt);
        }
    });

    jButtonOk.setText("OK");
    jButtonOk.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonOkActionPerformed(evt);
        }
    });

    javax.swing.GroupLayout jPanelSensorLayout = new javax.swing.GroupLayout(
        jPanelSensor);
    jPanelSensor.setLayout(jPanelSensorLayout);
    jPanelSensorLayout.setHorizontalGroup(
        jPanelSensorLayout.createParallelGroup(javax.swing.GroupLayout.
            Alignment.LEADING)
        .addGroup(jPanelSensorLayout.createSequentialGroup()
            .addContainerGap()
            .addGroup(jPanelSensorLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.LEADING)
```

```
                .addGroup(jPanelSensorLayout.createSequentialGroup()
                    .addComponent(jLabelId)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                        .RELATED)
                    .addComponent(jTextFieldId, javax.swing.GroupLayout.
                        PREFERRED_SIZE, 71, javax.swing.GroupLayout.
                        PREFERRED_SIZE))
                .addGroup(jPanelSensorLayout.createSequentialGroup()
                    .addComponent(jLabelType)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                        .RELATED)
                    .addComponent(jComboBoxType, javax.swing.GroupLayout.
                        PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                        javax.swing.GroupLayout.PREFERRED_SIZE)))
            .addContainerGap(99, Short.MAX_VALUE))
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
            jPanelSensorLayout.createSequentialGroup()
            .addContainerGap(39, Short.MAX_VALUE)
            .addComponent(jButtonOk, javax.swing.GroupLayout.PREFERRED_SIZE,
                73, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
                UNRELATED)
            .addComponent(jButtonCancel, javax.swing.GroupLayout.PREFERRED_SIZE
                , 73, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap())
    );
    jPanelSensorLayout.setVerticalGroup(
        jPanelSensorLayout.createParallelGroup(javax.swing.GroupLayout.
            Alignment.LEADING)
        .addGroup(jPanelSensorLayout.createSequentialGroup()
            .addContainerGap()
            .addGroup(jPanelSensorLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.BASELINE)
                .addComponent(jLabelId)
                .addComponent(jTextFieldId, javax.swing.GroupLayout.
                    PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                    .swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
                UNRELATED)
            .addGroup(jPanelSensorLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.BASELINE)
                .addComponent(jLabelType)
                .addComponent(jComboBoxType, javax.swing.GroupLayout.
                    PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                    .swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
                , 28, Short.MAX_VALUE)
            .addGroup(jPanelSensorLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.BASELINE)
                .addComponent(jButtonCancel)
                .addComponent(jButtonOk))
            .addContainerGap())
    );

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane
        ());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jPanelSensor, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addContainerGap())
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jPanelSensor, javax.swing.GroupLayout.DEFAULT_SIZE,
```

```
                    javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jButtonCancelActionPerformed(java.awt.event.ActionEvent evt) {//
        GEN-FIRST:event_jButtonCancelActionPerformed
        returnValue = JOptionPane.CANCEL_OPTION;
        this.setVisible(false);
    }//GEN-LAST:event_jButtonCancelActionPerformed

    private void jButtonOkActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
        FIRST:event_jButtonOkActionPerformed

        if(!jTextFieldId.getText().isEmpty()) {
            sensorId = jTextFieldId.getText();
            sensorType = (SensorType)jComboBoxType.getSelectedItem();
            returnValue = JOptionPane.OK_OPTION;
            this.setVisible(false);

        } else {
            JOptionPane.showMessageDialog(this,
                "No ID mentionned.",
                "Error",
                JOptionPane.ERROR_MESSAGE);
        }

    }//GEN-LAST:event_jButtonOkActionPerformed

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButtonCancel;
    private javax.swing.JButton jButtonOk;
    private javax.swing.JComboBox jComboBoxType;
    private javax.swing.JLabel jLabelId;
    private javax.swing.JLabel jLabelType;
    private javax.swing.JPanel jPanelSensor;
    private javax.swing.JTextField jTextFieldId;
    // End of variables declaration//GEN-END:variables
}
```

Listing H.3 – JDialogSensor Class

```
/*
 * JDialogSensorChooser.java
 *
 * Created on 17-mai-2012, 14:00:23
 */
package be.woine.thesis.dialog;

import be.woine.thesis.building.common.Building;
import be.woine.thesis.building.common.Floor;
import be.woine.thesis.building.common.Room;
import be.woine.thesis.model.BuildingTreeModel;
import be.woine.thesis.model.ValueSourceAbstractTableModel;
import be.woine.thesis.rule.common.ValueSource;
import java.util.ArrayList;
import javax.swing.JOptionPane;
import javax.swing.ListSelectionModel;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import javax.swing.tree.TreeSelectionModel;

/**
 *
 * @author Remy
 */
public class JDialogSensorChooser extends javax.swing.JDialog {
```

```
public Building building;

private ValueSource source;
private int returnValue = JOptionPane.CANCEL_OPTION;

/** Creates new form JDialogSensorChooser */
public JDialogSensorChooser(final Building building, java.awt.Frame parent,
    boolean modal) {

    super(parent, modal);
    this.building = building;

    /* Look And Feel */
    try {
        javax.swing.UIManager.setLookAndFeel(javax.swing.UIManager.
            getSystemLookAndFeelClassName());
    } catch (ClassNotFoundException ex) {
        System.out.println(ex.toString());
    } catch (InstantiationException ex) {
        System.out.println(ex.toString());
    } catch (IllegalAccessException ex) {
        System.out.println(ex.toString());
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
        System.out.println(ex.toString());
    }

    initComponents();

    jTableSensor.getSelectionModel().setSelectionMode(ListSelectionModel.
        SINGLE_SELECTION);

    jTreeBuilding.getSelectionModel().setSelectionMode(TreeSelectionModel.
        SINGLE_TREE_SELECTION);
    jTreeBuilding.getSelectionModel().addTreeSelectionListener(new
        TreeSelectionListener() {

        @Override
        public void valueChanged(TreeSelectionEvent e) {

            ArrayList<ValueSource> sensorList = new ArrayList<ValueSource>();

            try {
                if(jTreeBuilding.getLastSelectedPathComponent().getClass() ==
                    Room.class) {

                    for(int f = 0; f < building.getFloorCount(); f++) {
                        for(int r = 0; r < building.getFloor(f).getRoomCount();
                            r++) {
                            if(building.getFloor(f).getRoom(r) == e.getPath().
                                getLastPathComponent()) {
                                for(int s = 0; s < building.getFloor(f).getRoom
                                    (r).getSensorCount(); s++) {
                                    sensorList.add(new ValueSource(building, f,
                                        r, s));
                                }
                            }
                        }
                    }

                } else if(jTreeBuilding.getLastSelectedPathComponent().getClass
                    () == Floor.class) {

                    for(int f = 0; f < building.getFloorCount(); f++) {
                        if(building.getFloor(f) == e.getPath().
                            getLastPathComponent()) {
                            for(int r = 0; r < building.getFloor(f).
                                getRoomCount(); r++) {
                                for(int s = 0; s < building.getFloor(f).getRoom
                                    (r).getSensorCount(); s++)
```

```
                                    sensorList.add(new ValueSource(building, f,
                                        r, s));
                                }
                            }
                        }

                    } else if(jTreeBuilding.getLastSelectedPathComponent().getClass
                        () == Building.class) {

                        for(int f = 0; f < building.getFloorCount(); f++) {
                            for(int r = 0; r < building.getFloor(f).getRoomCount();
                                 r++) {
                                for(int s = 0; s < building.getFloor(f).getRoom(r).
                                    getSensorCount(); s++)
                                    sensorList.add(new ValueSource(building, f, r,
                                        s));
                            }
                        }

                    }

                    jTableSensor.setModel(new ValueSourceAbstractTableModel(
                        sensorList));

                } catch(NullPointerException ex) {
                    System.out.println(e.getSource() + "␣-->␣" + ex.toString());
                }
            }
        });
    }

    public int getReturnValue() {
        return returnValue;
    }

    public ValueSource getValueSource() {
        return source;
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:
        initComponents
    private void initComponents() {

        jPanelSensorSelector = new javax.swing.JPanel();
        jScrollPaneTree = new javax.swing.JScrollPane();
        jTreeBuilding = new javax.swing.JTree();
        jScrollPaneSensorTable = new javax.swing.JScrollPane();
        jTableSensor = new javax.swing.JTable();
        jLabelBuilding = new javax.swing.JLabel();
        jLabelSensorList = new javax.swing.JLabel();
        jButtonCancel = new javax.swing.JButton();
        jLabelSelectedSensor = new javax.swing.JLabel();
        jButtonOk = new javax.swing.JButton();
        jTextFieldSelectedSensor = new javax.swing.JTextField();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Sensor␣Chooser");

        jPanelSensorSelector.setBorder(javax.swing.BorderFactory.createTitledBorder
            (null, "Sensor␣Selector", javax.swing.border.TitledBorder.
            DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.DEFAULT_POSITION
            , new java.awt.Font("Tahoma", 1, 11), new java.awt.Color(0, 0, 255)));
            // NOI18N
```

```
jTreeBuilding.setModel(new BuildingTreeModel(building));
jScrollPaneTree.setViewportView(jTreeBuilding);

jTableSensor.setModel(new ValueSourceAbstractTableModel());
jTableSensor.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        jTableSensorMouseClicked(evt);
    }
});
jScrollPaneSensorTable.setViewportView(jTableSensor);

jLabelBuilding.setFont(new java.awt.Font("Tahoma", 1, 11));
jLabelBuilding.setText("Building :");

jLabelSensorList.setFont(new java.awt.Font("Tahoma", 1, 11));
jLabelSensorList.setText("Sensors' List :");

jButtonCancel.setText("Cancel");
jButtonCancel.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonCancelActionPerformed(evt);
    }
});

jLabelSelectedSensor.setText("Selected Sensor :");

jButtonOk.setText("OK");
jButtonOk.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonOkActionPerformed(evt);
    }
});

jTextFieldSelectedSensor.setEditable(false);

javax.swing.GroupLayout jPanelSensorSelectorLayout = new javax.swing.
    GroupLayout(jPanelSensorSelector);
jPanelSensorSelector.setLayout(jPanelSensorSelectorLayout);
jPanelSensorSelectorLayout.setHorizontalGroup(
    jPanelSensorSelectorLayout.createParallelGroup(javax.swing.GroupLayout.
        Alignment.LEADING)
    .addGroup(jPanelSensorSelectorLayout.createSequentialGroup()
        .addContainerGap()
        .addGroup(jPanelSensorSelectorLayout.createParallelGroup(javax.
            swing.GroupLayout.Alignment.LEADING)
            .addComponent(jScrollPaneTree, javax.swing.GroupLayout.
                PREFERRED_SIZE, 158, javax.swing.GroupLayout.PREFERRED_SIZE
                )
            .addComponent(jLabelBuilding))
        .addGap(18, 18, 18)
        .addGroup(jPanelSensorSelectorLayout.createParallelGroup(javax.
            swing.GroupLayout.Alignment.LEADING)
            .addComponent(jLabelSensorList)
            .addGroup(jPanelSensorSelectorLayout.createSequentialGroup()
                .addComponent(jScrollPaneSensorTable, javax.swing.
                    GroupLayout.PREFERRED_SIZE, 153, javax.swing.
                    GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)
                .addGroup(jPanelSensorSelectorLayout.createParallelGroup(
                    javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jLabelSelectedSensor)
                    .addComponent(jTextFieldSelectedSensor, javax.swing.
                        GroupLayout.DEFAULT_SIZE, 152, Short.MAX_VALUE)))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                jPanelSensorSelectorLayout.createSequentialGroup()
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                    .RELATED, 171, Short.MAX_VALUE)
                .addComponent(jButtonOk, javax.swing.GroupLayout.
                    PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                    PREFERRED_SIZE)
```

```
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                            .RELATED)
                        .addComponent(jButtonCancel, javax.swing.GroupLayout.
                            PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                            PREFERRED_SIZE)))
                .addContainerGap())
        );
        jPanelSensorSelectorLayout.setVerticalGroup(
            jPanelSensorSelectorLayout.createParallelGroup(javax.swing.GroupLayout.
                Alignment.LEADING)
            .addGroup(jPanelSensorSelectorLayout.createSequentialGroup()
                .addGroup(jPanelSensorSelectorLayout.createParallelGroup(javax.
                    swing.GroupLayout.Alignment.LEADING, false)
                    .addGroup(jPanelSensorSelectorLayout.createSequentialGroup()
                        .addContainerGap()
                        .addGroup(jPanelSensorSelectorLayout.createParallelGroup(
                            javax.swing.GroupLayout.Alignment.BASELINE)
                            .addComponent(jLabelBuilding)
                            .addComponent(jLabelSensorList))
                        .addGap(85, 85, 85)
                        .addComponent(jLabelSelectedSensor)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                            .RELATED)
                        .addComponent(jTextFieldSelectedSensor, javax.swing.
                            GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.
                            DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                            .RELATED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                            MAX_VALUE)
                        .addGroup(jPanelSensorSelectorLayout.createParallelGroup(
                            javax.swing.GroupLayout.Alignment.BASELINE)
                            .addComponent(jButtonCancel)
                            .addComponent(jButtonOk)))
                    .addGroup(jPanelSensorSelectorLayout.createSequentialGroup()
                        .addGap(31, 31, 31)
                        .addGroup(jPanelSensorSelectorLayout.createParallelGroup(
                            javax.swing.GroupLayout.Alignment.LEADING, false)
                            .addComponent(jScrollPaneTree, 0, 0, Short.MAX_VALUE)
                            .addComponent(jScrollPaneSensorTable, javax.swing.
                                GroupLayout.DEFAULT_SIZE, 242, Short.MAX_VALUE))))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE))
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane
            ());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelSensorSelector, javax.swing.GroupLayout.
                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelSensorSelector, javax.swing.GroupLayout.
                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jButtonOkActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
```

```
        FIRST:event_jButtonOkActionPerformed

        if((jTableSensor.getSelectedRow() != -1) && (!jTextFieldSelectedSensor.
            getText().isEmpty())) {
            returnValue = JOptionPane.OK_OPTION;
            this.setVisible(false);
        } else {
            JOptionPane.showMessageDialog(this,
                "No sensor selected.",
                "Error",
                JOptionPane.ERROR_MESSAGE);
        }

    }//GEN-LAST:event_jButtonOkActionPerformed

    private void jButtonCancelActionPerformed(java.awt.event.ActionEvent evt) {//
        GEN-FIRST:event_jButtonCancelActionPerformed
        returnValue = JOptionPane.CANCEL_OPTION;
        this.setVisible(false);
    }//GEN-LAST:event_jButtonCancelActionPerformed

    private void jTableSensorMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
        FIRST:event_jTableSensorMouseClicked

        String str = "";
        source = ((ValueSourceAbstractTableModel)jTableSensor.getModel()).
            getElementAt(jTableSensor.getSelectedRow());

        str += jTableSensor.getModel().getValueAt(jTableSensor.getSelectedRow(), 0)
            .toString();
        str += "@{";
        str += source.building.getFloor(source.floorIndex).getName();
        str += "_";
        str += source.building.getFloor(source.floorIndex).getRoom(source.roomIndex
            ).getName();
        str += "}";

        jTextFieldSelectedSensor.setText(str);
    }//GEN-LAST:event_jTableSensorMouseClicked

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButtonCancel;
    private javax.swing.JButton jButtonOk;
    private javax.swing.JLabel jLabelBuilding;
    private javax.swing.JLabel jLabelSelectedSensor;
    private javax.swing.JLabel jLabelSensorList;
    private javax.swing.JPanel jPanelSensorSelector;
    private javax.swing.JScrollPane jScrollPaneSensorTable;
    private javax.swing.JScrollPane jScrollPaneTree;
    private javax.swing.JTable jTableSensor;
    private javax.swing.JTextField jTextFieldSelectedSensor;
    private javax.swing.JTree jTreeBuilding;
    // End of variables declaration//GEN-END:variables
}
```

Listing H.4 – JDialogSensorChooser Class

```
/*
 * JDialogCompoundRule.java
 *
 * Created on 18-mai-2012, 12:56:17
 */
package be.woine.thesis.dialog;

import be.woine.thesis.building.common.Building;
import be.woine.thesis.model.SequenceRuleAbstractTableModel;
import be.woine.thesis.rule.common.CompoundRule;
import be.woine.thesis.rule.common.Rule;
import be.woine.thesis.rule.common.RuleType;
```

```
import javax.swing.JOptionPane;

/**
 *
 * @author Remy
 */
public class JDialogSequenceRule extends javax.swing.JDialog {

    private Building building;
    private Rule sequenceRule = new Rule();

    private int returnValue = JOptionPane.CANCEL_OPTION;

    /** Creates new form JDialogCompoundRule */
    public JDialogSequenceRule(Building building, java.awt.Frame parent, boolean
        modal) {
        super(parent, modal);

        this.building = building;

        /* Look And Feel */
        try {
            javax.swing.UIManager.setLookAndFeel(javax.swing.UIManager.
                getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException ex) {
            System.out.println(ex.toString());
        } catch (InstantiationException ex) {
            System.out.println(ex.toString());
        } catch (IllegalAccessException ex) {
            System.out.println(ex.toString());
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            System.out.println(ex.toString());
        }

        initComponents();
    }

    public int getReturnValue() {
        return returnValue;
    }

    public Rule getSequenceRule() {
        return sequenceRule;
    }


    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:
        initComponents
    private void initComponents() {

        jPanelSequenceRule = new javax.swing.JPanel();
        jScrollPaneRuleTable = new javax.swing.JScrollPane();
        jTableRule = new javax.swing.JTable();
        jButtonAdd = new javax.swing.JButton();
        jButtonEdit = new javax.swing.JButton();
        jButtonRemove = new javax.swing.JButton();
        jButtonCancel = new javax.swing.JButton();
        jButtonOk = new javax.swing.JButton();
        jLabelNumberOfSteps = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Sequence Rule");

        jPanelSequenceRule.setBorder(javax.swing.BorderFactory.createTitledBorder(
```

```
        null, "Sequence Rule", javax.swing.border.TitledBorder.
        DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.DEFAULT_POSITION
        , new java.awt.Font("Tahoma", 1, 11), new java.awt.Color(0, 0, 255)));
        // NOI18N

    jTableRule.setModel(new SequenceRuleAbstractTableModel());
    jScrollPaneRuleTable.setViewportView(jTableRule);

    jButtonAdd.setText("Add");
    jButtonAdd.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonAddActionPerformed(evt);
        }
    });

    jButtonEdit.setText("Edit");
    jButtonEdit.setEnabled(false);

    jButtonRemove.setText("Remove");
    jButtonRemove.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonRemoveActionPerformed(evt);
        }
    });

    jButtonCancel.setText("Cancel");
    jButtonCancel.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonCancelActionPerformed(evt);
        }
    });

    jButtonOk.setText("OK");
    jButtonOk.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonOkActionPerformed(evt);
        }
    });

    jLabelNumberOfSteps.setText("Number of steps = 0");

    javax.swing.GroupLayout jPanelSequenceRuleLayout = new javax.swing.
        GroupLayout(jPanelSequenceRule);
    jPanelSequenceRule.setLayout(jPanelSequenceRuleLayout);
    jPanelSequenceRuleLayout.setHorizontalGroup(
        jPanelSequenceRuleLayout.createParallelGroup(javax.swing.GroupLayout.
            Alignment.LEADING)
        .addGroup(jPanelSequenceRuleLayout.createSequentialGroup()
            .addContainerGap()
            .addGroup(jPanelSequenceRuleLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.LEADING)
                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                    jPanelSequenceRuleLayout.createSequentialGroup()
                    .addComponent(jLabelNumberOfSteps)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                        .RELATED, 228, Short.MAX_VALUE)
                    .addComponent(jButtonOk, javax.swing.GroupLayout.
                        PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                        PREFERRED_SIZE))
                .addComponent(jScrollPaneRuleTable, javax.swing.GroupLayout.
                    DEFAULT_SIZE, 400, Short.MAX_VALUE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
                UNRELATED)
            .addGroup(jPanelSequenceRuleLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.TRAILING)
                .addGroup(jPanelSequenceRuleLayout.createParallelGroup(javax.
                    swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jButtonAdd, javax.swing.GroupLayout.
                        PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                        PREFERRED_SIZE)
```

```
                        .addComponent(jButtonEdit, javax.swing.GroupLayout.
                            PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                            PREFERRED_SIZE)
                        .addComponent(jButtonRemove, javax.swing.GroupLayout.
                            PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                            PREFERRED_SIZE))
                    .addComponent(jButtonCancel, javax.swing.GroupLayout.
                        PREFERRED_SIZE, 73, javax.swing.GroupLayout.PREFERRED_SIZE)
                        )
                .addContainerGap())
        );
        jPanelSequenceRuleLayout.setVerticalGroup(
            jPanelSequenceRuleLayout.createParallelGroup(javax.swing.GroupLayout.
                Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                jPanelSequenceRuleLayout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanelSequenceRuleLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.LEADING)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                        jPanelSequenceRuleLayout.createSequentialGroup()
                        .addComponent(jButtonAdd)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                            .RELATED)
                        .addComponent(jButtonEdit)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                            .RELATED)
                        .addComponent(jButtonRemove)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                            .RELATED, 125, Short.MAX_VALUE)
                        .addComponent(jButtonCancel))
                    .addGroup(jPanelSequenceRuleLayout.createSequentialGroup()
                        .addComponent(jScrollPaneRuleTable, javax.swing.GroupLayout
                            .PREFERRED_SIZE, 192, javax.swing.GroupLayout.
                            PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                            .RELATED, 14, Short.MAX_VALUE)
                        .addGroup(jPanelSequenceRuleLayout.createParallelGroup(
                            javax.swing.GroupLayout.Alignment.BASELINE)
                            .addComponent(jButtonOk)
                            .addComponent(jLabelNumberOfSteps))))
                .addContainerGap())
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane
            ());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelSequenceRule, javax.swing.GroupLayout.
                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelSequenceRule, javax.swing.GroupLayout.
                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jButtonOkActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
```

```
        FIRST:event_jButtonOkActionPerformed

        if(jTableRule.getRowCount() > 0) {

            sequenceRule.setCompoundRuleList(((SequenceRuleAbstractTableModel)
                jTableRule.getModel()).getRuleList());

            returnValue = JOptionPane.OK_OPTION;
            setVisible(false);
        } else {
            JOptionPane.showMessageDialog(this,
                "No rule specified.",
                "Error",
                JOptionPane.ERROR_MESSAGE);
        }

    }//GEN-LAST:event_jButtonOkActionPerformed

    private void jButtonCancelActionPerformed(java.awt.event.ActionEvent evt) {//
        GEN-FIRST:event_jButtonCancelActionPerformed
        returnValue = JOptionPane.CANCEL_OPTION;
        setVisible(false);
    }//GEN-LAST:event_jButtonCancelActionPerformed

    private void jButtonAddActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
        FIRST:event_jButtonAddActionPerformed

        JDialogRuleChooser ruleDialog = new JDialogRuleChooser(null, true);
        ruleDialog.setLocationByPlatform(true);
        ruleDialog.jRadioButtonSequence.setEnabled(false);
        ruleDialog.jComboBoxActionType.setEnabled(false);
        ruleDialog.setVisible(true);

        if(ruleDialog.getReturnValue() == JOptionPane.OK_OPTION) {
            if(ruleDialog.getRuleType() == RuleType.FIXED) {

                JDialogSimpleRuleFixed fixedDialog = new JDialogSimpleRuleFixed(
                    building, null, true);
                fixedDialog.setLocationByPlatform(true);
                fixedDialog.setVisible(true);

                if(fixedDialog.getReturnValue() == JOptionPane.OK_OPTION) {
                    CompoundRule compoundRule = new CompoundRule();
                    compoundRule.addSimpleRule(fixedDialog.getSimpleRule());
                    ((SequenceRuleAbstractTableModel)jTableRule.getModel()).addRule
                        (compoundRule);
                }

            } else if(ruleDialog.getRuleType() == RuleType.RANGE) {

                JDialogSimpleRuleRange rangeDialog = new JDialogSimpleRuleRange(
                    building, null, true);
                rangeDialog.setLocationByPlatform(true);
                rangeDialog.setVisible(true);

                if(rangeDialog.getReturnValue() == JOptionPane.OK_OPTION) {
                    CompoundRule compoundRule = new CompoundRule();
                    compoundRule.addSimpleRule(rangeDialog.getSimpleRule());
                    ((SequenceRuleAbstractTableModel)jTableRule.getModel()).addRule
                        (compoundRule);
                }

            } else if(ruleDialog.getRuleType() == RuleType.COMPOUND) {

                JDialogCompoundRule compoundDialog = new JDialogCompoundRule(
                    building, null, true);
                compoundDialog.setLocationByPlatform(true);
                compoundDialog.setVisible(true);

                if(compoundDialog.getReturnValue() == JOptionPane.OK_OPTION) {
```

```
                    ((SequenceRuleAbstractTableModel)jTableRule.getModel()).addRule
                        (compoundDialog.getCompoundRule());
                }

            }
        }

        jLabelNumberOfSteps.setText("Number of steps = " + jTableRule.getRowCount()
            );

    }//GEN-LAST:event_jButtonAddActionPerformed

    private void jButtonRemoveActionPerformed(java.awt.event.ActionEvent evt) {//
        GEN-FIRST:event_jButtonRemoveActionPerformed
        if(jTableRule.getSelectedRow() >= 0)
            ((SequenceRuleAbstractTableModel)jTableRule.getModel()).removeRule(
                jTableRule.getSelectedRow());

        jLabelNumberOfSteps.setText("Number of steps = " + jTableRule.getRowCount()
            );
    }//GEN-LAST:event_jButtonRemoveActionPerformed

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButtonAdd;
    private javax.swing.JButton jButtonCancel;
    private javax.swing.JButton jButtonEdit;
    private javax.swing.JButton jButtonOk;
    private javax.swing.JButton jButtonRemove;
    private javax.swing.JLabel jLabelNumberOfSteps;
    private javax.swing.JPanel jPanelSequenceRule;
    private javax.swing.JScrollPane jScrollPaneRuleTable;
    private javax.swing.JTable jTableRule;
    // End of variables declaration//GEN-END:variables
}
```

Listing H.5 – JDialogSequenceRule Class

```
/*
 * JDialogSimpleRuleFixed.java
 *
 * Created on 17-mai-2012, 12:39:04
 */
package be.woine.thesis.dialog;

import be.woine.thesis.building.common.Building;
import be.woine.thesis.rule.common.IntegerRuleBody;
import be.woine.thesis.rule.common.RuleOperator;
import be.woine.thesis.rule.common.SimpleRule;
import be.woine.thesis.rule.common.StringRuleBody;
import be.woine.thesis.rule.common.ValueSource;
import javax.swing.JOptionPane;

/**
 *
 * @author Remy
 */
public class JDialogSimpleRuleFixed extends javax.swing.JDialog {

    private Building building;
    private SimpleRule simpleRule = new SimpleRule();

    private RuleOperator[] operatorInteger = {RuleOperator.SMALLER_THAN,
                                              RuleOperator.
                                                  SMALLER_THAN_OR_EQUAL_TO,
                                              RuleOperator.GREATER_THAN,
                                              RuleOperator.
                                                  GREATER_THAN_OR_EQUAL_TO,
                                              RuleOperator.EQUAL_TO,
                                              RuleOperator.NOT_EQUAL_TO};
```

```
    private RuleOperator[] operatorString = {RuleOperator.EQUAL_TO,
                                             RuleOperator.NOT_EQUAL_TO};

    private int returnValue = JOptionPane.CANCEL_OPTION;


    /** Creates new form JDialogSimpleRuleFixed */
    public JDialogSimpleRuleFixed(Building building, java.awt.Frame parent, boolean
         modal) {

        super(parent, modal);
        this.building = building;

        /* Look And Feel */
        try {
            javax.swing.UIManager.setLookAndFeel(javax.swing.UIManager.
                getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException ex) {
            System.out.println(ex.toString());
        } catch (InstantiationException ex) {
            System.out.println(ex.toString());
        } catch (IllegalAccessException ex) {
            System.out.println(ex.toString());
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            System.out.println(ex.toString());
        }

        initComponents();

        jComboBoxValueClass.removeAllItems();
        jComboBoxValueClass.addItem("Integer");
        jComboBoxValueClass.addItem("String");

    }

    public int getReturnValue() {
        return returnValue;
    }

    public SimpleRule getSimpleRule() {
        return simpleRule;
    }


    private void updateString() {
        String str = "";
        if(simpleRule.getSource() != null) {
            str += "(";
            str += simpleRule.getSource().building.getFloor(simpleRule.getSource().
                floorIndex).getRoom(simpleRule.getSource().roomIndex).getSensor(
                simpleRule.getSource().sensorIndex).getType().getVar();
            str += "@";
            str += simpleRule.getSource().building.getFloor(simpleRule.getSource().
                floorIndex).getName();
            str += "_";
            str += simpleRule.getSource().building.getFloor(simpleRule.getSource().
                floorIndex).getRoom(simpleRule.getSource().roomIndex).getName();
            str += ")";
        }
        jTextFieldString.setText(str + jComboBoxOperator.getSelectedItem() +
            jTextFieldValue.getText());
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
```

```
// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:
    initComponents
private void initComponents() {

    jPanelFixedValue = new javax.swing.JPanel();
    jLabelSensorType = new javax.swing.JLabel();
    jButtonGetSensor = new javax.swing.JButton();
    jComboBoxOperator = new javax.swing.JComboBox();
    jTextFieldValue = new javax.swing.JTextField();
    jComboBoxValueClass = new javax.swing.JComboBox();
    jTextFieldString = new javax.swing.JTextField();
    jButtonCancel = new javax.swing.JButton();
    jButtonOk = new javax.swing.JButton();

    setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
    setTitle("Simple Fixed Rule");

    jPanelFixedValue.setBorder(javax.swing.BorderFactory.createTitledBorder(
        null, "Fixed Rule", javax.swing.border.TitledBorder.
        DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.DEFAULT_POSITION
        , new java.awt.Font("Tahoma", 1, 11), new java.awt.Color(0, 0, 255)));
        // NOI18N

    jLabelSensorType.setText("<sensor-type>");

    jButtonGetSensor.setText("Get");
    jButtonGetSensor.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonGetSensorActionPerformed(evt);
        }
    });

    jComboBoxOperator.setModel(new javax.swing.DefaultComboBoxModel(new String
        [] { "Item 1", "Item 2", "Item 3", "Item 4" }));
    jComboBoxOperator.addItemListener(new java.awt.event.ItemListener() {
        public void itemStateChanged(java.awt.event.ItemEvent evt) {
            jComboBoxOperatorItemStateChanged(evt);
        }
    });

    jTextFieldValue.addCaretListener(new javax.swing.event.CaretListener() {
        public void caretUpdate(javax.swing.event.CaretEvent evt) {
            jTextFieldValueCaretUpdate(evt);
        }
    });

    jComboBoxValueClass.setModel(new javax.swing.DefaultComboBoxModel(new
        String[] { "Item 1", "Item 2", "Item 3", "Item 4" }));
    jComboBoxValueClass.addItemListener(new java.awt.event.ItemListener() {
        public void itemStateChanged(java.awt.event.ItemEvent evt) {
            jComboBoxValueClassItemStateChanged(evt);
        }
    });

    jTextFieldString.setEditable(false);

    jButtonCancel.setText("Cancel");
    jButtonCancel.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonCancelActionPerformed(evt);
        }
    });

    jButtonOk.setText("OK");
    jButtonOk.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonOkActionPerformed(evt);
        }
    });
```

```
javax.swing.GroupLayout jPanelFixedValueLayout = new javax.swing.
    GroupLayout(jPanelFixedValue);
jPanelFixedValue.setLayout(jPanelFixedValueLayout);
jPanelFixedValueLayout.setHorizontalGroup(
    jPanelFixedValueLayout.createParallelGroup(javax.swing.GroupLayout.
        Alignment.LEADING)
    .addGroup(jPanelFixedValueLayout.createSequentialGroup()
        .addContainerGap()
        .addGroup(jPanelFixedValueLayout.createParallelGroup(javax.swing.
            GroupLayout.Alignment.LEADING)
            .addGroup(jPanelFixedValueLayout.createParallelGroup(javax.
                swing.GroupLayout.Alignment.TRAILING, false)
                .addComponent(jTextFieldString)
                .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
                    jPanelFixedValueLayout.createSequentialGroup()
                    .addComponent(jLabelSensorType, javax.swing.GroupLayout
                        .PREFERRED_SIZE, 83, javax.swing.GroupLayout.
                        PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.
                        ComponentPlacement.UNRELATED)
                    .addComponent(jComboBoxOperator, javax.swing.
                        GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout
                        .DEFAULT_SIZE, javax.swing.GroupLayout.
                        PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.
                        ComponentPlacement.RELATED)
                    .addComponent(jTextFieldValue, javax.swing.GroupLayout.
                        PREFERRED_SIZE, 72, javax.swing.GroupLayout.
                        PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.
                        ComponentPlacement.RELATED)
                    .addComponent(jComboBoxValueClass, javax.swing.
                        GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout
                        .DEFAULT_SIZE, javax.swing.GroupLayout.
                        PREFERRED_SIZE)))
            .addGroup(jPanelFixedValueLayout.createSequentialGroup()
                .addGap(13, 13, 13)
                .addComponent(jButtonGetSensor))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                jPanelFixedValueLayout.createSequentialGroup()
                .addComponent(jButtonOk, javax.swing.GroupLayout.
                    PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                    PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                    .RELATED)
                .addComponent(jButtonCancel, javax.swing.GroupLayout.
                    PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                    PREFERRED_SIZE)))
        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.
            MAX_VALUE))
);
jPanelFixedValueLayout.setVerticalGroup(
    jPanelFixedValueLayout.createParallelGroup(javax.swing.GroupLayout.
        Alignment.LEADING)
    .addGroup(jPanelFixedValueLayout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jButtonGetSensor)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
            )
        .addGroup(jPanelFixedValueLayout.createParallelGroup(javax.swing.
            GroupLayout.Alignment.BASELINE)
            .addComponent(jLabelSensorType)
            .addComponent(jComboBoxOperator, javax.swing.GroupLayout.
                PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                .swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jTextFieldValue, javax.swing.GroupLayout.
                PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                .swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jComboBoxValueClass, javax.swing.GroupLayout.
                PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
```

```
                        .swing.GroupLayout.PREFERRED_SIZE))
                .addGap(33, 33, 33)
                .addComponent(jTextFieldString, javax.swing.GroupLayout.
                    PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.
                    swing.GroupLayout.PREFERRED_SIZE)
                .addGap(27, 27, 27)
                .addGroup(jPanelFixedValueLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.BASELINE)
                    .addComponent(jButtonCancel)
                    .addComponent(jButtonOk))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE))
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane
            ());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelFixedValue, javax.swing.GroupLayout.
                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelFixedValue, javax.swing.GroupLayout.
                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jComboBoxValueClassItemStateChanged(java.awt.event.ItemEvent evt)
        {//GEN-FIRST:event_jComboBoxValueClassItemStateChanged

        if(jComboBoxValueClass.getSelectedItem() != null) {
            if(((String)jComboBoxValueClass.getSelectedItem()).compareTo("String")
                == 0) {
                jComboBoxOperator.removeAllItems();
                for(int i = 0; i < operatorString.length; i++)
                    jComboBoxOperator.addItem(operatorString[i].getSymbol());
            } else if(((String)jComboBoxValueClass.getSelectedItem()).compareTo("
                Integer") == 0) {
                jComboBoxOperator.removeAllItems();
                for(int i = 0; i < operatorInteger.length; i++)
                    jComboBoxOperator.addItem(operatorInteger[i].getSymbol());
            }
        }

        updateString();

    }//GEN-LAST:event_jComboBoxValueClassItemStateChanged

    private void jButtonOkActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
        FIRST:event_jButtonOkActionPerformed

        if(simpleRule.getSource() == null) {
            JOptionPane.showMessageDialog(this,
                "No sensor selected.",
                "Error",
                JOptionPane.ERROR_MESSAGE);
        } else {
            if(jTextFieldValue.getText().isEmpty()) {
```

```
                JOptionPane.showMessageDialog(this,
                    "No value entered.",
                    "Error",
                    JOptionPane.ERROR_MESSAGE);
            } else {
                if(((String)jComboBoxValueClass.getSelectedItem()).compareTo("
                    Integer") == 0) {

                    try {
                        Integer value = Integer.parseInt(jTextFieldValue.getText())
                            ;

                        simpleRule.addRuleBody(new IntegerRuleBody(value,
                            operatorInteger[jComboBoxOperator.getSelectedIndex()]))
                            ;

                        returnValue = JOptionPane.OK_OPTION;
                        setVisible(false);
                    } catch (java.lang.NumberFormatException ex) {
                        JOptionPane.showMessageDialog(this,
                            "Value entered is not an Integer.",
                            "Error",
                            JOptionPane.ERROR_MESSAGE);
                    }

                } else {
                    simpleRule.addRuleBody(new StringRuleBody(jTextFieldValue.
                        getText(), operatorString[jComboBoxOperator.
                        getSelectedIndex()]));
                    returnValue = JOptionPane.OK_OPTION;
                    setVisible(false);
                }
            }
        }

    }//GEN-LAST:event_jButtonOkActionPerformed

    private void jButtonCancelActionPerformed(java.awt.event.ActionEvent evt) {//
        GEN-FIRST:event_jButtonCancelActionPerformed
        returnValue = JOptionPane.CANCEL_OPTION;
        setVisible(false);
    }//GEN-LAST:event_jButtonCancelActionPerformed

    private void jComboBoxOperatorItemStateChanged(java.awt.event.ItemEvent evt) {
        //GEN-FIRST:event_jComboBoxOperatorItemStateChanged
        updateString();
    }//GEN-LAST:event_jComboBoxOperatorItemStateChanged

    private void jTextFieldValueCaretUpdate(javax.swing.event.CaretEvent evt) {//
        GEN-FIRST:event_jTextFieldValueCaretUpdate
        updateString();
    }//GEN-LAST:event_jTextFieldValueCaretUpdate

    private void jButtonGetSensorActionPerformed(java.awt.event.ActionEvent evt) {
        //GEN-FIRST:event_jButtonGetSensorActionPerformed

        JDialogSensorChooser dialog = new JDialogSensorChooser(building, null, true
            );
        dialog.setLocationByPlatform(true);
        dialog.setVisible(true);

        if(dialog.getReturnValue() == JOptionPane.OK_OPTION) {
            ValueSource source = dialog.getValueSource();

            simpleRule.setSource(source);
            jLabelSensorType.setText(source.building.getFloor(source.floorIndex).
                getRoom(source.roomIndex).getSensor(source.sensorIndex).getType().
                toString());
        }
```

```
        updateString();
    }//GEN-LAST:event_jButtonGetSensorActionPerformed

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButtonCancel;
    private javax.swing.JButton jButtonGetSensor;
    private javax.swing.JButton jButtonOk;
    private javax.swing.JComboBox jComboBoxOperator;
    private javax.swing.JComboBox jComboBoxValueClass;
    private javax.swing.JLabel jLabelSensorType;
    private javax.swing.JPanel jPanelFixedValue;
    private javax.swing.JTextField jTextFieldString;
    private javax.swing.JTextField jTextFieldValue;
    // End of variables declaration//GEN-END:variables
}
```

Listing H.6 – JDialogSimpleRuleFixed Class

```
/*
 * JDialogSimpleRuleFixed.java
 *
 * Created on 17-mai-2012, 12:39:04
 */
package be.woine.thesis.dialog;

import be.woine.thesis.building.common.Building;
import be.woine.thesis.rule.common.IntegerRuleBody;
import be.woine.thesis.rule.common.RuleOperator;
import be.woine.thesis.rule.common.SimpleRule;
import be.woine.thesis.rule.common.ValueSource;
import javax.swing.JOptionPane;

/**
 *
 * @author Remy
 */
public class JDialogSimpleRuleRange extends javax.swing.JDialog {

    private Building building;
    private SimpleRule simpleRule = new SimpleRule();

    private RuleOperator[] operator = {RuleOperator.INCLUDED_IN,
                                       RuleOperator.NOT_INCLUDED_IN};


    private int returnValue = JOptionPane.CANCEL_OPTION;


    /** Creates new form JDialogSimpleRuleFixed */
    public JDialogSimpleRuleRange(Building building, java.awt.Frame parent, boolean
        modal) {

        super(parent, modal);
        this.building = building;

        /* Look And Feel */
        try {
            javax.swing.UIManager.setLookAndFeel(javax.swing.UIManager.
                getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException ex) {
            System.out.println(ex.toString());
        } catch (InstantiationException ex) {
            System.out.println(ex.toString());
        } catch (IllegalAccessException ex) {
            System.out.println(ex.toString());
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            System.out.println(ex.toString());
        }
```

```
        initComponents();

        jComboBoxOperator.removeAllItems();
        for(int o = 0; o < operator.length; o++)
            jComboBoxOperator.addItem(operator[o].getSymbol());


    }

    public int getReturnValue() {
        return returnValue;
    }

    public SimpleRule getSimpleRule() {
        return simpleRule;
    }


    private void updateString() {
        String str = "";
        if(simpleRule.getSource() != null) {
            str += "(";
            str += simpleRule.getSource().building.getFloor(simpleRule.getSource().
                floorIndex).getRoom(simpleRule.getSource().roomIndex).getSensor(
                simpleRule.getSource().sensorIndex).getType().getVar();
            str += "@";
            str += simpleRule.getSource().building.getFloor(simpleRule.getSource().
                floorIndex).getName();
            str += "_";
            str += simpleRule.getSource().building.getFloor(simpleRule.getSource().
                floorIndex).getRoom(simpleRule.getSource().roomIndex).getName();
            str += ")";
        }

        str += jComboBoxOperator.getSelectedItem();

        if(jCheckBoxIncludeMinValue.isSelected())
            str += "[";
        else
            str += "]";

        str += jSpinnerMinValue.getValue();
        str += ":";
        str += jSpinnerMaxValue.getValue();

        if(jCheckBoxIncludeMaxValue.isSelected())
            str += "]";
        else
            str += "[";

        jTextFieldString.setText(str);
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:
        initComponents
    private void initComponents() {

        jPanelRangeValue = new javax.swing.JPanel();
        jLabelSensorType = new javax.swing.JLabel();
        jButtonGetSensor = new javax.swing.JButton();
        jComboBoxOperator = new javax.swing.JComboBox();
        jTextFieldString = new javax.swing.JTextField();
        jButtonCancel = new javax.swing.JButton();
        jButtonOk = new javax.swing.JButton();
```

```
jCheckBoxIncludeMinValue = new javax.swing.JCheckBox();
jSpinnerMinValue = new javax.swing.JSpinner();
jSpinnerMaxValue = new javax.swing.JSpinner();
jCheckBoxIncludeMaxValue = new javax.swing.JCheckBox();
jLabelFrom = new javax.swing.JLabel();
jLabelTo = new javax.swing.JLabel();
jLabelInRange = new javax.swing.JLabel();

setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
setTitle("Simple Range Rule");

jPanelRangeValue.setBorder(javax.swing.BorderFactory.createTitledBorder(
    null, "Range Rule", javax.swing.border.TitledBorder.
    DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.DEFAULT_POSITION
    , new java.awt.Font("Tahoma", 1, 11), new java.awt.Color(0, 0, 255)));
    // NOI18N

jLabelSensorType.setText("<sensor-type>");

jButtonGetSensor.setText("Get");
jButtonGetSensor.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonGetSensorActionPerformed(evt);
    }
});

jComboBoxOperator.setModel(new javax.swing.DefaultComboBoxModel(new String
    [] { "Item 1", "Item 2", "Item 3", "Item 4" }));
jComboBoxOperator.addItemListener(new java.awt.event.ItemListener() {
    public void itemStateChanged(java.awt.event.ItemEvent evt) {
        jComboBoxOperatorItemStateChanged(evt);
    }
});

jTextFieldString.setEditable(false);

jButtonCancel.setText("Cancel");
jButtonCancel.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonCancelActionPerformed(evt);
    }
});

jButtonOk.setText("OK");
jButtonOk.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonOkActionPerformed(evt);
    }
});

jCheckBoxIncludeMinValue.setSelected(true);
jCheckBoxIncludeMinValue.setText("included");
jCheckBoxIncludeMinValue.addChangeListener(new javax.swing.event.
    ChangeListener() {
    public void stateChanged(javax.swing.event.ChangeEvent evt) {
        jCheckBoxIncludeMinValueStateChanged(evt);
    }
});

jSpinnerMinValue.setModel(new javax.swing.SpinnerNumberModel());
jSpinnerMinValue.addChangeListener(new javax.swing.event.ChangeListener() {
    public void stateChanged(javax.swing.event.ChangeEvent evt) {
        jSpinnerMinValueStateChanged(evt);
    }
});

jSpinnerMaxValue.setModel(new javax.swing.SpinnerNumberModel(Integer.
    valueOf(1), null, null, Integer.valueOf(1)));
jSpinnerMaxValue.addChangeListener(new javax.swing.event.ChangeListener() {
    public void stateChanged(javax.swing.event.ChangeEvent evt) {
```

```
            jSpinnerMaxValueStateChanged(evt);
        }
    });

    jCheckBoxIncludeMaxValue.setSelected(true);
    jCheckBoxIncludeMaxValue.setText("included");
    jCheckBoxIncludeMaxValue.addChangeListener(new javax.swing.event.
        ChangeListener() {
        public void stateChanged(javax.swing.event.ChangeEvent evt) {
            jCheckBoxIncludeMaxValueStateChanged(evt);
        }
    });

    jLabelFrom.setText("from :");

    jLabelTo.setText("to :");

    jLabelInRange.setText("in range : ");

    javax.swing.GroupLayout jPanelRangeValueLayout = new javax.swing.
        GroupLayout(jPanelRangeValue);
    jPanelRangeValue.setLayout(jPanelRangeValueLayout);
    jPanelRangeValueLayout.setHorizontalGroup(
        jPanelRangeValueLayout.createParallelGroup(javax.swing.GroupLayout.
            Alignment.LEADING)
        .addGroup(jPanelRangeValueLayout.createSequentialGroup()
            .addGroup(jPanelRangeValueLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.LEADING)
                .addGroup(jPanelRangeValueLayout.createSequentialGroup()
                    .addContainerGap()
                    .addGroup(jPanelRangeValueLayout.createParallelGroup(javax.
                        swing.GroupLayout.Alignment.LEADING)
                        .addGroup(jPanelRangeValueLayout.createSequentialGroup
                            ()
                            .addComponent(jLabelSensorType, javax.swing.
                                GroupLayout.PREFERRED_SIZE, 83, javax.swing.
                                GroupLayout.PREFERRED_SIZE)
                            .addPreferredGap(javax.swing.LayoutStyle.
                                ComponentPlacement.UNRELATED)
                            .addComponent(jComboBoxOperator, javax.swing.
                                GroupLayout.PREFERRED_SIZE, javax.swing.
                                GroupLayout.DEFAULT_SIZE, javax.swing.
                                GroupLayout.PREFERRED_SIZE)
                            .addPreferredGap(javax.swing.LayoutStyle.
                                ComponentPlacement.UNRELATED)
                            .addComponent(jLabelInRange))
                        .addGroup(jPanelRangeValueLayout.createSequentialGroup
                            ()
                            .addGap(13, 13, 13)
                            .addComponent(jButtonGetSensor))
                        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                            jPanelRangeValueLayout.createSequentialGroup()
                            .addComponent(jButtonOk, javax.swing.GroupLayout.
                                PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                                PREFERRED_SIZE)
                            .addPreferredGap(javax.swing.LayoutStyle.
                                ComponentPlacement.RELATED)
                            .addComponent(jButtonCancel, javax.swing.
                                GroupLayout.PREFERRED_SIZE, 73, javax.swing.
                                GroupLayout.PREFERRED_SIZE))
                        .addComponent(jTextFieldString, javax.swing.GroupLayout
                            .Alignment.TRAILING, javax.swing.GroupLayout.
                            DEFAULT_SIZE, 289, Short.MAX_VALUE)))
                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                    jPanelRangeValueLayout.createSequentialGroup()
                    .addGap(133, 133, 133)
                    .addGroup(jPanelRangeValueLayout.createParallelGroup(javax.
                        swing.GroupLayout.Alignment.TRAILING)
                        .addComponent(jLabelFrom)
                        .addComponent(jLabelTo))
```

```
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                        .UNRELATED)
                    .addGroup(jPanelRangeValueLayout.createParallelGroup(javax.
                        swing.GroupLayout.Alignment.LEADING)
                        .addGroup(jPanelRangeValueLayout.createSequentialGroup
                            ()
                            .addComponent(jSpinnerMinValue, javax.swing.
                                GroupLayout.DEFAULT_SIZE, 60, Short.MAX_VALUE)
                            .addPreferredGap(javax.swing.LayoutStyle.
                                ComponentPlacement.RELATED)
                            .addComponent(jCheckBoxIncludeMinValue))
                        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                            jPanelRangeValueLayout.createSequentialGroup()
                            .addComponent(jSpinnerMaxValue, javax.swing.
                                GroupLayout.DEFAULT_SIZE, 60, Short.MAX_VALUE)
                            .addPreferredGap(javax.swing.LayoutStyle.
                                ComponentPlacement.RELATED)
                            .addComponent(jCheckBoxIncludeMaxValue)))))
            .addContainerGap())
    );
    jPanelRangeValueLayout.setVerticalGroup(
        jPanelRangeValueLayout.createParallelGroup(javax.swing.GroupLayout.
            Alignment.LEADING)
        .addGroup(jPanelRangeValueLayout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jButtonGetSensor)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
                )
            .addGroup(jPanelRangeValueLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.BASELINE)
                .addComponent(jLabelSensorType)
                .addComponent(jComboBoxOperator, javax.swing.GroupLayout.
                    PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                    .swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabelInRange))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
                UNRELATED)
            .addGroup(jPanelRangeValueLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.BASELINE)
                .addComponent(jCheckBoxIncludeMinValue)
                .addComponent(jSpinnerMinValue, javax.swing.GroupLayout.
                    PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                    .swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabelFrom))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
                )
            .addGroup(jPanelRangeValueLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.BASELINE)
                .addComponent(jCheckBoxIncludeMaxValue)
                .addComponent(jSpinnerMaxValue, javax.swing.GroupLayout.
                    PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                    .swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabelTo))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
                , 19, Short.MAX_VALUE)
            .addComponent(jTextFieldString, javax.swing.GroupLayout.
                PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.
                swing.GroupLayout.PREFERRED_SIZE)
            .addGap(30, 30, 30)
            .addGroup(jPanelRangeValueLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.BASELINE)
                .addComponent(jButtonCancel)
                .addComponent(jButtonOk))
            .addContainerGap())
    );

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane
        ());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
```

```
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelRangeValue, javax.swing.GroupLayout.
                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelRangeValue, javax.swing.GroupLayout.
                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jButtonOkActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
        FIRST:event_jButtonOkActionPerformed

        if(simpleRule.getSource() == null) {
            JOptionPane.showMessageDialog(this,
                "No sensor selected.",
                "Error",
                JOptionPane.ERROR_MESSAGE);
        } else {

            RuleOperator minRuleOp, maxRuleOp;
            if(jCheckBoxIncludeMinValue.isSelected())
                minRuleOp = RuleOperator.GREATER_THAN_OR_EQUAL_TO;
            else
                minRuleOp = RuleOperator.GREATER_THAN;

            if(jCheckBoxIncludeMaxValue.isSelected())
                maxRuleOp = RuleOperator.SMALLER_THAN_OR_EQUAL_TO;
            else
                maxRuleOp = RuleOperator.SMALLER_THAN;

            simpleRule.setSpecialOperator(operator[jComboBoxOperator.
                getSelectedIndex()]);
            simpleRule.addRuleBody(new IntegerRuleBody(Integer.valueOf(
                jSpinnerMinValue.getValue().toString()), minRuleOp));
            simpleRule.addRuleBody(new IntegerRuleBody(Integer.valueOf(
                jSpinnerMaxValue.getValue().toString()), maxRuleOp));

            returnValue = JOptionPane.OK_OPTION;
            setVisible(false);

        }

    }//GEN-LAST:event_jButtonOkActionPerformed

    private void jButtonCancelActionPerformed(java.awt.event.ActionEvent evt) {//
        GEN-FIRST:event_jButtonCancelActionPerformed
        returnValue = JOptionPane.CANCEL_OPTION;
        setVisible(false);
    }//GEN-LAST:event_jButtonCancelActionPerformed

    private void jComboBoxOperatorItemStateChanged(java.awt.event.ItemEvent evt) {
        //GEN-FIRST:event_jComboBoxOperatorItemStateChanged
        updateString();
    }//GEN-LAST:event_jComboBoxOperatorItemStateChanged

    private void jButtonGetSensorActionPerformed(java.awt.event.ActionEvent evt) {
        //GEN-FIRST:event_jButtonGetSensorActionPerformed
```

```
        JDialogSensorChooser dialog = new JDialogSensorChooser(building, null, true
            );
        dialog.setLocationByPlatform(true);
        dialog.setVisible(true);

        if(dialog.getReturnValue() == JOptionPane.OK_OPTION) {
            ValueSource source = dialog.getValueSource();

            simpleRule.setSource(source);
            jLabelSensorType.setText(source.building.getFloor(source.floorIndex).
                getRoom(source.roomIndex).getSensor(source.sensorIndex).getType().
                toString());
        }

        updateString();
    }//GEN-LAST:event_jButtonGetSensorActionPerformed

    private void jCheckBoxIncludeMinValueStateChanged(javax.swing.event.ChangeEvent
         evt) {//GEN-FIRST:event_jCheckBoxIncludeMinValueStateChanged
        updateString();
    }//GEN-LAST:event_jCheckBoxIncludeMinValueStateChanged

    private void jCheckBoxIncludeMaxValueStateChanged(javax.swing.event.ChangeEvent
         evt) {//GEN-FIRST:event_jCheckBoxIncludeMaxValueStateChanged
        updateString();
    }//GEN-LAST:event_jCheckBoxIncludeMaxValueStateChanged

    private void jSpinnerMinValueStateChanged(javax.swing.event.ChangeEvent evt) {
        //GEN-FIRST:event_jSpinnerMinValueStateChanged
        if(Integer.valueOf(jSpinnerMinValue.getValue().toString()) >= Integer.
            valueOf(jSpinnerMaxValue.getValue().toString()))
            jSpinnerMinValue.setValue(Integer.valueOf(jSpinnerMaxValue.getValue().
                toString()) - 1);
        updateString();
    }//GEN-LAST:event_jSpinnerMinValueStateChanged

    private void jSpinnerMaxValueStateChanged(javax.swing.event.ChangeEvent evt) {
        //GEN-FIRST:event_jSpinnerMaxValueStateChanged
        if(Integer.valueOf(jSpinnerMaxValue.getValue().toString()) <= Integer.
            valueOf(jSpinnerMinValue.getValue().toString()))
            jSpinnerMaxValue.setValue(Integer.valueOf(jSpinnerMinValue.getValue().
                toString()) + 1);
        updateString();
    }//GEN-LAST:event_jSpinnerMaxValueStateChanged

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButtonCancel;
    private javax.swing.JButton jButtonGetSensor;
    private javax.swing.JButton jButtonOk;
    private javax.swing.JCheckBox jCheckBoxIncludeMaxValue;
    private javax.swing.JCheckBox jCheckBoxIncludeMinValue;
    private javax.swing.JComboBox jComboBoxOperator;
    private javax.swing.JLabel jLabelFrom;
    private javax.swing.JLabel jLabelInRange;
    private javax.swing.JLabel jLabelSensorType;
    private javax.swing.JLabel jLabelTo;
    private javax.swing.JPanel jPanelRangeValue;
    private javax.swing.JSpinner jSpinnerMaxValue;
    private javax.swing.JSpinner jSpinnerMinValue;
    private javax.swing.JTextField jTextFieldString;
    // End of variables declaration//GEN-END:variables
}
```

Listing H.7 – JDialogSimpleRuleRange Class

```
/*
 * JDialogTreeElement.java
 *
 * Created on 18-mai-2012, 16:37:13
```

```
 */
package be.woine.thesis.dialog;

import javax.swing.JOptionPane;
import javax.swing.border.TitledBorder;

/**
 *
 * @author Remy
 */
public class JDialogTreeElement extends javax.swing.JDialog {

    private String elementName = "";
    private int returnValue = JOptionPane.CANCEL_OPTION;

    /** Creates new form JDialogTreeElement */
    public JDialogTreeElement(String element, java.awt.Frame parent, boolean modal)
        {
        super(parent, modal);

        /* Look And Feel */
        try {
            javax.swing.UIManager.setLookAndFeel(javax.swing.UIManager.
                getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException ex) {
            System.out.println(ex.toString());
        } catch (InstantiationException ex) {
            System.out.println(ex.toString());
        } catch (IllegalAccessException ex) {
            System.out.println(ex.toString());
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            System.out.println(ex.toString());
        }

        initComponents();
        ((TitledBorder)jPanelElement.getBorder()).setTitle(element);
    }

    public int getReturnValue() {
        return returnValue;
    }

    public String getElementName() {
        return elementName;
    }



    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:
        initComponents
    private void initComponents() {

        jPanelElement = new javax.swing.JPanel();
        jLabelName = new javax.swing.JLabel();
        jTextFieldName = new javax.swing.JTextField();
        jButtonCancel = new javax.swing.JButton();
        jButtonOk = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);

        jPanelElement.setBorder(javax.swing.BorderFactory.createTitledBorder(null,
            "Element", javax.swing.border.TitledBorder.DEFAULT_JUSTIFICATION, javax
            .swing.border.TitledBorder.DEFAULT_POSITION, new java.awt.Font("Tahoma"
            , 1, 11), new java.awt.Color(0, 0, 255))); // NOI18N
```

```
jLabelName.setText("Name␣:␣");

jButtonCancel.setText("Cancel");
jButtonCancel.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonCancelActionPerformed(evt);
    }
});

jButtonOk.setText("OK");
jButtonOk.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonOkActionPerformed(evt);
    }
});

javax.swing.GroupLayout jPanelElementLayout = new javax.swing.GroupLayout(
    jPanelElement);
jPanelElement.setLayout(jPanelElementLayout);
jPanelElementLayout.setHorizontalGroup(
    jPanelElementLayout.createParallelGroup(javax.swing.GroupLayout.
        Alignment.LEADING)
    .addGroup(jPanelElementLayout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jLabelName)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
            )
        .addComponent(jTextFieldName, javax.swing.GroupLayout.
            PREFERRED_SIZE, 90, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addContainerGap(75, Short.MAX_VALUE))
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
        jPanelElementLayout.createSequentialGroup()
        .addContainerGap(50, Short.MAX_VALUE)
        .addComponent(jButtonOk, javax.swing.GroupLayout.PREFERRED_SIZE,
            73, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
            UNRELATED)
        .addComponent(jButtonCancel, javax.swing.GroupLayout.PREFERRED_SIZE
            , 73, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addContainerGap())
);
jPanelElementLayout.setVerticalGroup(
    jPanelElementLayout.createParallelGroup(javax.swing.GroupLayout.
        Alignment.LEADING)
    .addGroup(jPanelElementLayout.createSequentialGroup()
        .addContainerGap()
        .addGroup(jPanelElementLayout.createParallelGroup(javax.swing.
            GroupLayout.Alignment.BASELINE)
            .addComponent(jLabelName)
            .addComponent(jTextFieldName, javax.swing.GroupLayout.
                PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                .swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
            , 18, Short.MAX_VALUE)
        .addGroup(jPanelElementLayout.createParallelGroup(javax.swing.
            GroupLayout.Alignment.BASELINE)
            .addComponent(jButtonCancel)
            .addComponent(jButtonOk))
        .addContainerGap())
);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane
    ());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jPanelElement, javax.swing.GroupLayout.DEFAULT_SIZE,
```

```
                    javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelElement, javax.swing.GroupLayout.DEFAULT_SIZE,
                    javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jButtonCancelActionPerformed(java.awt.event.ActionEvent evt) {//
        GEN-FIRST:event_jButtonCancelActionPerformed
        returnValue = JOptionPane.CANCEL_OPTION;
        this.setVisible(false);
    }//GEN-LAST:event_jButtonCancelActionPerformed

    private void jButtonOkActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
        FIRST:event_jButtonOkActionPerformed

        if(!jTextFieldName.getText().isEmpty()) {

            elementName = jTextFieldName.getText();
            returnValue = JOptionPane.OK_OPTION;
            this.setVisible(false);

        } else {

            JOptionPane.showMessageDialog(this,
                "No name specified.",
                "Error",
                JOptionPane.ERROR_MESSAGE);

        }

    }//GEN-LAST:event_jButtonOkActionPerformed

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButtonCancel;
    private javax.swing.JButton jButtonOk;
    private javax.swing.JLabel jLabelName;
    private javax.swing.JPanel jPanelElement;
    private javax.swing.JTextField jTextFieldName;
    // End of variables declaration//GEN-END:variables
}
```

Listing H.8 – JDialogTreeElement Class

```
/*
 * JDialogAlertProcessing.java
 *
 * Created on 26-mai-2012, 15:09:19
 */
package be.woine.thesis.dialog;

/**
 *
 * @author Remy
 */
public class JDialogAlertProcessing extends javax.swing.JDialog {

    /** Creates new form JDialogAlertProcessing */
    public JDialogAlertProcessing(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
```

```
        /* Look And Feel */
        try {
            javax.swing.UIManager.setLookAndFeel(javax.swing.UIManager.
                getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException ex) {
            System.out.println(ex.toString());
        } catch (InstantiationException ex) {
            System.out.println(ex.toString());
        } catch (IllegalAccessException ex) {
            System.out.println(ex.toString());
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            System.out.println(ex.toString());
        }

        initComponents();
    }

    public void setRuleId(int id) {
        jTextFieldRuleId.setText(String.valueOf(id));
    }

    public void setRuleType(String type) {
        jTextFieldRuleType.setText(type);
    }

    public void setBuildingName(String buildingName) {
        jTextFieldBuilding.setText(buildingName);
    }

    public void setTime(String time) {
        jTextFieldTime.setText(time);
    }

    public void setDescription(String desc) {
        jTextAreaDescription.setText(desc);
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:
        initComponents
    private void initComponents() {

        jPanelAlertDialog = new javax.swing.JPanel();
        jLabelRuleId = new javax.swing.JLabel();
        jTextFieldRuleId = new javax.swing.JTextField();
        jLabelRuleType = new javax.swing.JLabel();
        jTextFieldRuleType = new javax.swing.JTextField();
        jLabelBuilding = new javax.swing.JLabel();
        jLabelDescription = new javax.swing.JLabel();
        jTextFieldBuilding = new javax.swing.JTextField();
        jScrollPaneDescription = new javax.swing.JScrollPane();
        jTextAreaDescription = new javax.swing.JTextArea();
        jLabelTime = new javax.swing.JLabel();
        jTextFieldTime = new javax.swing.JTextField();
        jButtonClose = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Alert␣Dialog");
        setResizable(false);

        jPanelAlertDialog.setBorder(javax.swing.BorderFactory.createTitledBorder(
            null, "Alert␣Dialog", javax.swing.border.TitledBorder.
            DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.DEFAULT_POSITION
            , new java.awt.Font("Tahoma", 1, 11), new java.awt.Color(255, 0, 0)));
            // NOI18N
```

```
jLabelRuleId.setText("Rule ID : ");

jTextFieldRuleId.setEditable(false);
jTextFieldRuleId.setHorizontalAlignment(javax.swing.JTextField.CENTER);
jTextFieldRuleId.setText("0");

jLabelRuleType.setText("Rule Type : ");

jTextFieldRuleType.setEditable(false);
jTextFieldRuleType.setHorizontalAlignment(javax.swing.JTextField.CENTER);
jTextFieldRuleType.setText("<type>");

jLabelBuilding.setText("Building : ");

jLabelDescription.setText("Description : ");

jTextFieldBuilding.setEditable(false);
jTextFieldBuilding.setHorizontalAlignment(javax.swing.JTextField.CENTER);
jTextFieldBuilding.setText("<building>");

jTextAreaDescription.setColumns(20);
jTextAreaDescription.setEditable(false);
jTextAreaDescription.setFont(new java.awt.Font("Tahoma", 0, 11)); // NOI18N
jTextAreaDescription.setRows(5);
jScrollPaneDescription.setViewportView(jTextAreaDescription);

jLabelTime.setText("Time : ");

jTextFieldTime.setEditable(false);
jTextFieldTime.setHorizontalAlignment(javax.swing.JTextField.CENTER);
jTextFieldTime.setText("<time>");

jButtonClose.setText("Close");
jButtonClose.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonCloseActionPerformed(evt);
    }
});

javax.swing.GroupLayout jPanelAlertDialogLayout = new javax.swing.
    GroupLayout(jPanelAlertDialog);
jPanelAlertDialog.setLayout(jPanelAlertDialogLayout);
jPanelAlertDialogLayout.setHorizontalGroup(
    jPanelAlertDialogLayout.createParallelGroup(javax.swing.GroupLayout.
        Alignment.LEADING)
    .addGroup(jPanelAlertDialogLayout.createSequentialGroup()
        .addGroup(jPanelAlertDialogLayout.createParallelGroup(javax.swing.
            GroupLayout.Alignment.LEADING)
            .addGroup(jPanelAlertDialogLayout.createSequentialGroup()
                .addGap(15, 15, 15)
                .addGroup(jPanelAlertDialogLayout.createParallelGroup(javax
                    .swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(jLabelRuleId)
                    .addComponent(jLabelRuleType)
                    .addComponent(jLabelBuilding)
                    .addComponent(jLabelTime))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                    .RELATED)
                .addGroup(jPanelAlertDialogLayout.createParallelGroup(javax
                    .swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jTextFieldRuleId, javax.swing.GroupLayout
                        .DEFAULT_SIZE, 281, Short.MAX_VALUE)
                    .addComponent(jTextFieldRuleType, javax.swing.
                        GroupLayout.DEFAULT_SIZE, 281, Short.MAX_VALUE)
                    .addComponent(jTextFieldBuilding, javax.swing.
                        GroupLayout.DEFAULT_SIZE, 281, Short.MAX_VALUE)
                    .addComponent(jTextFieldTime, javax.swing.GroupLayout.
                        DEFAULT_SIZE, 281, Short.MAX_VALUE)))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
```

```
                        jPanelAlertDialogLayout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(jLabelDescription)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                            .RELATED)
                        .addComponent(jScrollPaneDescription, javax.swing.
                            GroupLayout.DEFAULT_SIZE, 281, Short.MAX_VALUE))
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                        jPanelAlertDialogLayout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(jButtonClose, javax.swing.GroupLayout.
                            PREFERRED_SIZE, 73, javax.swing.GroupLayout.
                            PREFERRED_SIZE)))
                .addContainerGap())
        );
        jPanelAlertDialogLayout.setVerticalGroup(
            jPanelAlertDialogLayout.createParallelGroup(javax.swing.GroupLayout.
                Alignment.LEADING)
            .addGroup(jPanelAlertDialogLayout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanelAlertDialogLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabelRuleId)
                    .addComponent(jTextFieldRuleId, javax.swing.GroupLayout.
                        PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                        .swing.GroupLayout.PREFERRED_SIZE))
                .addGap(8, 8, 8)
                .addGroup(jPanelAlertDialogLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.BASELINE)
                    .addComponent(jTextFieldRuleType, javax.swing.GroupLayout.
                        PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                        .swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabelRuleType))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
                    UNRELATED)
                .addGroup(jPanelAlertDialogLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabelBuilding)
                    .addComponent(jTextFieldBuilding, javax.swing.GroupLayout.
                        PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                        .swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
                    UNRELATED)
                .addGroup(jPanelAlertDialogLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.BASELINE)
                    .addComponent(jTextFieldTime, javax.swing.GroupLayout.
                        PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                        .swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabelTime))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
                    UNRELATED)
                .addGroup(jPanelAlertDialogLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.LEADING)
                    .addComponent(jLabelDescription)
                    .addComponent(jScrollPaneDescription, javax.swing.GroupLayout.
                        DEFAULT_SIZE, 157, Short.MAX_VALUE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
                    UNRELATED)
                .addComponent(jButtonClose)
                .addContainerGap())
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane
            ());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelAlertDialog, javax.swing.GroupLayout.
```

```
                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelAlertDialog, javax.swing.GroupLayout.
                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jButtonCloseActionPerformed(java.awt.event.ActionEvent evt) {//GEN
        -FIRST:event_jButtonCloseActionPerformed
        this.setVisible(false);
    }//GEN-LAST:event_jButtonCloseActionPerformed

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButtonClose;
    private javax.swing.JLabel jLabelBuilding;
    private javax.swing.JLabel jLabelDescription;
    private javax.swing.JLabel jLabelRuleId;
    private javax.swing.JLabel jLabelRuleType;
    private javax.swing.JLabel jLabelTime;
    private javax.swing.JPanel jPanelAlertDialog;
    private javax.swing.JScrollPane jScrollPaneDescription;
    private javax.swing.JTextArea jTextAreaDescription;
    private javax.swing.JTextField jTextFieldBuilding;
    private javax.swing.JTextField jTextFieldRuleId;
    private javax.swing.JTextField jTextFieldRuleType;
    private javax.swing.JTextField jTextFieldTime;
    // End of variables declaration//GEN-END:variables
}
```

Listing H.9 – JDialogAlertProcessing Class

```
/*
 * JDialogEmulateSensor.java
 *
 * Created on 27-mai-2012, 15:20:55
 */
package be.woine.thesis.dialog;

import alice.logictuple.LogicTuple;
import alice.logictuple.Value;
import alice.tucson.api.AgentId;
import alice.tucson.api.ContextNotAvailableException;
import alice.tucson.api.InvalidTupleCentreIdException;
import alice.tucson.api.OperationNotAllowedException;
import alice.tucson.api.Tucson;
import alice.tucson.api.TucsonContextSynch;
import alice.tucson.api.TupleCentreId;
import alice.tucson.api.UnreachableNodeException;
import be.woine.thesis.building.common.SensorType;
import be.woine.thesis.rule.common.ValueSource;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;
import java.util.concurrent.atomic.AtomicInteger;
import javax.swing.JOptionPane;
import javax.swing.Timer;

/**
 *
```

```
 * @author Remy
 */
public class JDialogEmulateSensor extends javax.swing.JDialog {

    private TucsonContextSynch ctx;
    private TupleCentreId tcid;
    private SensorType type;
    private String tupleCentreId = "";
    private String tupleName = "";
    private String sensorId = "";
    private Timer emulatedSensor;
    private static final AtomicInteger uniqueId = new AtomicInteger();

    /** Creates new form JDialogEmulateSensor */
    public JDialogEmulateSensor(ValueSource source, java.awt.Frame parent, boolean
        modal) {

        super(parent, modal);

        /* Look And Feel */
        try {
            javax.swing.UIManager.setLookAndFeel(javax.swing.UIManager.
                getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException ex) {
            System.out.println(ex.toString());
        } catch (InstantiationException ex) {
            System.out.println(ex.toString());
        } catch (IllegalAccessException ex) {
            System.out.println(ex.toString());
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            System.out.println(ex.toString());
        }

        initComponents();

        jTextFieldBuilding.setText(source.building.getName());
        jTextFieldFloor.setText(source.building.getFloor(source.floorIndex).getName
            ());
        jTextFieldRoom.setText(source.building.getFloor(source.floorIndex).getRoom(
            source.roomIndex).getName());

        tupleCentreId = source.building.getFloor(source.floorIndex).getName() + "_"
             + source.building.getFloor(source.floorIndex).getRoom(source.roomIndex
            ).getName();
        tupleName = source.building.getFloor(source.floorIndex).getRoom(source.
            roomIndex).getSensor(source.sensorIndex).getType().toString();
        type = source.building.getFloor(source.floorIndex).getRoom(source.roomIndex
            ).getSensor(source.sensorIndex).getType();
        sensorId = source.building.getFloor(source.floorIndex).getRoom(source.
            roomIndex).getSensor(source.sensorIndex).getId();

        jTextFieldSensorType.setText(tupleName);

        try {
            ctx = new TucsonContextSynch(Tucson.getContext(new AgentId("emulator_"
                + uniqueId.incrementAndGet())));
            tcid = new TupleCentreId(tupleCentreId);
        } catch (ContextNotAvailableException ex) {
            jTextFieldStatus.setText(ex.toString());
            System.err.println(ex);
        } catch (InvalidTupleCentreIdException ex) {
            jTextFieldStatus.setText(ex.toString());
            System.err.println(ex);
        }

    }

    private void sendValue(Object value) {
        try {
            if(value.getClass() == Integer.class) {
```

```
                ctx.out(tcid, new LogicTuple(tupleName, new Value(sensorId), new
                    Value((Integer)value)));
                jTextFieldStatus.setText("Value '" + value + "' sent to tuple
                    centre '" + tupleCentreId + "'.");
            } else {
                ctx.out(tcid, new LogicTuple(tupleName, new Value(sensorId), new
                    Value((String)value)));
                jTextFieldStatus.setText("Value '" + value + "' sent to tuple
                    centre '" + tupleCentreId + "'.");
            }
        } catch (OperationNotAllowedException ex) {
            jTextFieldStatus.setText(ex.toString());
            System.err.println(ex);
        } catch (UnreachableNodeException ex) {
            jTextFieldStatus.setText(ex.toString());
            System.err.println(ex);
        }
    }

    private Timer getEmulatedSensor(final int delay) {

        ActionListener action = new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {

                int value = 0;

                if(type.toString().compareTo(SensorType.TEMPERATURE.toString()) ==
                    0)
                    value = ((new Random().nextInt(201)) - 50);
                else if(type.toString().compareTo(SensorType.HUMIDITY.toString())
                    == 0)
                    value = (new Random().nextInt(101));
                else if(type.toString().compareTo(SensorType.LUMINOSITY.toString())
                     == 0)
                    value = (new Random().nextInt(20001));
                else if(type.toString().compareTo(SensorType.PRESSURE.toString())
                    == 0)
                    value = (new Random().nextInt(1901) + 100);
                else
                    value = 0;

                sendValue(value);
            }
        };

        return new Timer(delay, action);
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:
        initComponents
    private void initComponents() {

        buttonGroup = new javax.swing.ButtonGroup();
        jPanelSensorEmulator = new javax.swing.JPanel();
        jLabelBuilding = new javax.swing.JLabel();
        jLabelFloor = new javax.swing.JLabel();
        jLabelRoom = new javax.swing.JLabel();
        jTextFieldBuilding = new javax.swing.JTextField();
        jTextFieldFloor = new javax.swing.JTextField();
        jTextFieldRoom = new javax.swing.JTextField();
        jPanelSendValue = new javax.swing.JPanel();
        jRadioButtonSingle = new javax.swing.JRadioButton();
```

```
        jRadioButtonSequence = new javax.swing.JRadioButton();
        jTextFieldSingleValue = new javax.swing.JTextField();
        jComboBoxSingleValueType = new javax.swing.JComboBox();
        jButtonSendSingleValue = new javax.swing.JButton();
        jLabelValue = new javax.swing.JLabel();
        jLabelPeriod = new javax.swing.JLabel();
        jSpinnerPeriod = new javax.swing.JSpinner();
        jLabelSecond = new javax.swing.JLabel();
        jButtonStartStopSequence = new javax.swing.JButton();
        jButtonExit = new javax.swing.JButton();
        jPanelStatus = new javax.swing.JPanel();
        jTextFieldStatus = new javax.swing.JTextField();
        jLabelSensorType = new javax.swing.JLabel();
        jTextFieldSensorType = new javax.swing.JTextField();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("Sensor Emulator");
        setResizable(false);

        jPanelSensorEmulator.setBorder(javax.swing.BorderFactory.createTitledBorder
            (null, "Emulate Sensor", javax.swing.border.TitledBorder.
            DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.DEFAULT_POSITION
            , new java.awt.Font("Tahoma", 1, 11), new java.awt.Color(0, 0, 255)));
            // NOI18N

        jLabelBuilding.setText("Building :");

        jLabelFloor.setText("Floor :");

        jLabelRoom.setText("Room :");

        jTextFieldBuilding.setEditable(false);
        jTextFieldBuilding.setHorizontalAlignment(javax.swing.JTextField.CENTER);

        jTextFieldFloor.setEditable(false);
        jTextFieldFloor.setHorizontalAlignment(javax.swing.JTextField.CENTER);

        jTextFieldRoom.setEditable(false);
        jTextFieldRoom.setHorizontalAlignment(javax.swing.JTextField.CENTER);

        jPanelSendValue.setBorder(javax.swing.BorderFactory.createTitledBorder(null
            , "Send Value", javax.swing.border.TitledBorder.DEFAULT_JUSTIFICATION,
            javax.swing.border.TitledBorder.DEFAULT_POSITION, new java.awt.Font("
            Tahoma", 1, 11))); // NOI18N

        buttonGroup.add(jRadioButtonSingle);
        jRadioButtonSingle.setSelected(true);
        jRadioButtonSingle.setText("One single value");
        jRadioButtonSingle.addItemListener(new java.awt.event.ItemListener() {
            public void itemStateChanged(java.awt.event.ItemEvent evt) {
                jRadioButtonSingleItemStateChanged(evt);
            }
        });

        buttonGroup.add(jRadioButtonSequence);
        jRadioButtonSequence.setText("Random sequence");
        jRadioButtonSequence.addItemListener(new java.awt.event.ItemListener() {
            public void itemStateChanged(java.awt.event.ItemEvent evt) {
                jRadioButtonSequenceItemStateChanged(evt);
            }
        });

        jTextFieldSingleValue.setHorizontalAlignment(javax.swing.JTextField.CENTER)
            ;

        jComboBoxSingleValueType.setModel(new javax.swing.DefaultComboBoxModel(new
            String[] { "Integer", "String" }));

        jButtonSendSingleValue.setText("Send");
        jButtonSendSingleValue.addActionListener(new java.awt.event.ActionListener
```

```
        () {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonSendSingleValueActionPerformed(evt);
        }
    });

    jLabelValue.setText("Value :");

    jLabelPeriod.setText("Period :");

    jSpinnerPeriod.setModel(new javax.swing.SpinnerNumberModel(Integer.valueOf
        (1), Integer.valueOf(1), null, Integer.valueOf(1)));
    jSpinnerPeriod.setEnabled(false);

    jLabelSecond.setText("[s]");

    jButtonStartStopSequence.setText("Start");
    jButtonStartStopSequence.setEnabled(false);
    jButtonStartStopSequence.addActionListener(new java.awt.event.
        ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonStartStopSequenceActionPerformed(evt);
        }
    });

    javax.swing.GroupLayout jPanelSendValueLayout = new javax.swing.GroupLayout
        (jPanelSendValue);
    jPanelSendValue.setLayout(jPanelSendValueLayout);
    jPanelSendValueLayout.setHorizontalGroup(
        jPanelSendValueLayout.createParallelGroup(javax.swing.GroupLayout.
            Alignment.LEADING)
        .addGroup(jPanelSendValueLayout.createSequentialGroup()
            .addContainerGap()
            .addGroup(jPanelSendValueLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.LEADING)
                .addComponent(jRadioButtonSingle)
                .addComponent(jRadioButtonSequence))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
                )
            .addGroup(jPanelSendValueLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.LEADING)
                .addGroup(jPanelSendValueLayout.createSequentialGroup()
                    .addGap(2, 2, 2)
                    .addComponent(jLabelValue)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                        .RELATED)
                    .addGroup(jPanelSendValueLayout.createParallelGroup(javax.
                        swing.GroupLayout.Alignment.TRAILING, false)
                        .addComponent(jButtonSendSingleValue, javax.swing.
                            GroupLayout.Alignment.LEADING, javax.swing.
                            GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.
                            DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(jTextFieldSingleValue, javax.swing.
                            GroupLayout.Alignment.LEADING, javax.swing.
                            GroupLayout.PREFERRED_SIZE, 77, javax.swing.
                            GroupLayout.PREFERRED_SIZE))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                        .RELATED)
                    .addComponent(jComboBoxSingleValueType, javax.swing.
                        GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.
                        DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGroup(jPanelSendValueLayout.createSequentialGroup()
                    .addComponent(jLabelPeriod)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                        .RELATED)
                    .addGroup(jPanelSendValueLayout.createParallelGroup(javax.
                        swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jButtonStartStopSequence, javax.swing.
                            GroupLayout.PREFERRED_SIZE, 77, javax.swing.
                            GroupLayout.PREFERRED_SIZE)
```

```
                        .addComponent(jSpinnerPeriod, javax.swing.GroupLayout.
                            DEFAULT_SIZE, 85, Short.MAX_VALUE))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                        .RELATED)
                    .addComponent(jLabelSecond)
                    .addGap(48, 48, 48)))
            .addContainerGap())
);
jPanelSendValueLayout.setVerticalGroup(
    jPanelSendValueLayout.createParallelGroup(javax.swing.GroupLayout.
        Alignment.LEADING)
    .addGroup(jPanelSendValueLayout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jRadioButtonSingle)
        .addGap(58, 58, 58)
        .addComponent(jRadioButtonSequence)
        .addContainerGap(62, Short.MAX_VALUE))
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
        jPanelSendValueLayout.createSequentialGroup()
        .addContainerGap(32, Short.MAX_VALUE)
        .addGroup(jPanelSendValueLayout.createParallelGroup(javax.swing.
            GroupLayout.Alignment.BASELINE)
            .addComponent(jLabelValue)
            .addComponent(jTextFieldSingleValue, javax.swing.GroupLayout.
                PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                .swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jComboBoxSingleValueType, javax.swing.GroupLayout
                .PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
            )
        .addComponent(jButtonSendSingleValue)
        .addGap(32, 32, 32)
        .addGroup(jPanelSendValueLayout.createParallelGroup(javax.swing.
            GroupLayout.Alignment.BASELINE)
            .addComponent(jLabelPeriod)
            .addComponent(jSpinnerPeriod, javax.swing.GroupLayout.
                PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                .swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jLabelSecond))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
            )
        .addComponent(jButtonStartStopSequence)
        .addContainerGap())
);

jButtonExit.setIcon(new javax.swing.ImageIcon(getClass().getResource("/be/
    woine/thesis/utilz/exit.png"))); // NOI18N
jButtonExit.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonExitActionPerformed(evt);
    }
});

jPanelStatus.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "
    Status", javax.swing.border.TitledBorder.DEFAULT_JUSTIFICATION, javax.
    swing.border.TitledBorder.DEFAULT_POSITION, new java.awt.Font("Tahoma",
     1, 11))); // NOI18N

jTextFieldStatus.setEditable(false);

javax.swing.GroupLayout jPanelStatusLayout = new javax.swing.GroupLayout(
    jPanelStatus);
jPanelStatus.setLayout(jPanelStatusLayout);
jPanelStatusLayout.setHorizontalGroup(
    jPanelStatusLayout.createParallelGroup(javax.swing.GroupLayout.
        Alignment.LEADING)
    .addGroup(jPanelStatusLayout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jTextFieldStatus, javax.swing.GroupLayout.
```

```
                DEFAULT_SIZE, 290, Short.MAX_VALUE)
            .addContainerGap())
    );
    jPanelStatusLayout.setVerticalGroup(
        jPanelStatusLayout.createParallelGroup(javax.swing.GroupLayout.
            Alignment.LEADING)
        .addGroup(jPanelStatusLayout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jTextFieldStatus, javax.swing.GroupLayout.
                PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.
                swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                MAX_VALUE))
    );

    jLabelSensorType.setText("Type_:");

    jTextFieldSensorType.setEditable(false);
    jTextFieldSensorType.setHorizontalAlignment(javax.swing.JTextField.CENTER);

    javax.swing.GroupLayout jPanelSensorEmulatorLayout = new javax.swing.
        GroupLayout(jPanelSensorEmulator);
    jPanelSensorEmulator.setLayout(jPanelSensorEmulatorLayout);
    jPanelSensorEmulatorLayout.setHorizontalGroup(
        jPanelSensorEmulatorLayout.createParallelGroup(javax.swing.GroupLayout.
            Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
            jPanelSensorEmulatorLayout.createSequentialGroup()
            .addContainerGap()
            .addGroup(jPanelSensorEmulatorLayout.createParallelGroup(javax.
                swing.GroupLayout.Alignment.TRAILING)
                .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
                    jPanelSensorEmulatorLayout.createSequentialGroup()
                    .addGroup(jPanelSensorEmulatorLayout.createParallelGroup(
                        javax.swing.GroupLayout.Alignment.TRAILING)
                        .addComponent(jLabelRoom)
                        .addComponent(jLabelFloor)
                        .addComponent(jLabelBuilding)
                        .addComponent(jLabelSensorType))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                        .RELATED)
                    .addGroup(jPanelSensorEmulatorLayout.createParallelGroup(
                        javax.swing.GroupLayout.Alignment.LEADING)
                        .addGroup(jPanelSensorEmulatorLayout.
                            createSequentialGroup()
                            .addGroup(jPanelSensorEmulatorLayout.
                                createParallelGroup(javax.swing.GroupLayout.
                                Alignment.LEADING)
                                .addComponent(jTextFieldFloor, javax.swing.
                                    GroupLayout.Alignment.TRAILING, javax.swing
                                    .GroupLayout.PREFERRED_SIZE, 110, javax.
                                    swing.GroupLayout.PREFERRED_SIZE)
                                .addComponent(jTextFieldBuilding, javax.swing.
                                    GroupLayout.Alignment.TRAILING, javax.swing
                                    .GroupLayout.PREFERRED_SIZE, 110, javax.
                                    swing.GroupLayout.PREFERRED_SIZE)
                                .addComponent(jTextFieldRoom, javax.swing.
                                    GroupLayout.Alignment.TRAILING, javax.swing
                                    .GroupLayout.PREFERRED_SIZE, 110, javax.
                                    swing.GroupLayout.PREFERRED_SIZE))
                            .addPreferredGap(javax.swing.LayoutStyle.
                                ComponentPlacement.RELATED, 105, Short.
                                MAX_VALUE)
                            .addComponent(jButtonExit, javax.swing.GroupLayout.
                                PREFERRED_SIZE, 60, javax.swing.GroupLayout.
                                PREFERRED_SIZE))
                        .addComponent(jTextFieldSensorType, javax.swing.
                            GroupLayout.PREFERRED_SIZE, 110, javax.swing.
                            GroupLayout.PREFERRED_SIZE)))
                .addComponent(jPanelStatus, javax.swing.GroupLayout.Alignment.
```

```
                    LEADING, javax.swing.GroupLayout.PREFERRED_SIZE, javax.
                    swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.
                    PREFERRED_SIZE)
                .addComponent(jPanelSendValue, javax.swing.GroupLayout.
                    Alignment.LEADING, javax.swing.GroupLayout.PREFERRED_SIZE,
                    javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.
                    GroupLayout.PREFERRED_SIZE))
            .addContainerGap())
);
jPanelSensorEmulatorLayout.setVerticalGroup(
    jPanelSensorEmulatorLayout.createParallelGroup(javax.swing.GroupLayout.
        Alignment.LEADING)
    .addGroup(jPanelSensorEmulatorLayout.createSequentialGroup()
        .addContainerGap()
        .addGroup(jPanelSensorEmulatorLayout.createParallelGroup(javax.
            swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanelSensorEmulatorLayout.createSequentialGroup()
                .addGroup(jPanelSensorEmulatorLayout.createParallelGroup(
                    javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jTextFieldBuilding, javax.swing.
                        GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout
                        .DEFAULT_SIZE, javax.swing.GroupLayout.
                        PREFERRED_SIZE)
                    .addComponent(jLabelBuilding))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                    .UNRELATED)
                .addGroup(jPanelSensorEmulatorLayout.createParallelGroup(
                    javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jTextFieldFloor, javax.swing.GroupLayout.
                        PREFERRED_SIZE, javax.swing.GroupLayout.
                        DEFAULT_SIZE, javax.swing.GroupLayout.
                        PREFERRED_SIZE)
                    .addComponent(jLabelFloor))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                    .UNRELATED)
                .addGroup(jPanelSensorEmulatorLayout.createParallelGroup(
                    javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jTextFieldRoom, javax.swing.GroupLayout.
                        PREFERRED_SIZE, javax.swing.GroupLayout.
                        DEFAULT_SIZE, javax.swing.GroupLayout.
                        PREFERRED_SIZE)
                    .addComponent(jLabelRoom)))
            .addComponent(jButtonExit, javax.swing.GroupLayout.
                PREFERRED_SIZE, 60, javax.swing.GroupLayout.PREFERRED_SIZE)
                )
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
            UNRELATED)
        .addGroup(jPanelSensorEmulatorLayout.createParallelGroup(javax.
            swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabelSensorType)
            .addComponent(jTextFieldSensorType, javax.swing.GroupLayout.
                PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                .swing.GroupLayout.PREFERRED_SIZE))
        .addGap(18, 18, 18)
        .addComponent(jPanelSendValue, javax.swing.GroupLayout.
            PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.
            swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
            UNRELATED)
        .addComponent(jPanelStatus, javax.swing.GroupLayout.PREFERRED_SIZE,
             javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.
            PREFERRED_SIZE)
        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.
            MAX_VALUE))
);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane
    ());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
```

```
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelSensorEmulator, javax.swing.GroupLayout.
                    PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.
                    swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelSensorEmulator, javax.swing.GroupLayout.
                    PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.
                    swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jButtonStartStopSequenceActionPerformed(java.awt.event.ActionEvent
         evt) {//GEN-FIRST:event_jButtonStartStopSequenceActionPerformed

        if(jButtonStartStopSequence.getText().compareTo("Start") == 0) {
            jButtonStartStopSequence.setText("Stop");

            emulatedSensor = getEmulatedSensor(((Integer)jSpinnerPeriod.getValue())
                *1000);
            emulatedSensor.start();

        } else {
            jButtonStartStopSequence.setText("Start");
            emulatedSensor.stop();
        }

    }//GEN-LAST:event_jButtonStartStopSequenceActionPerformed

    private void jRadioButtonSingleItemStateChanged(java.awt.event.ItemEvent evt) {
        //GEN-FIRST:event_jRadioButtonSingleItemStateChanged

        if(jRadioButtonSingle.isSelected()) {
            jTextFieldSingleValue.setEnabled(true);
            jComboBoxSingleValueType.setEnabled(true);
            jButtonSendSingleValue.setEnabled(true);
        } else {
            jTextFieldSingleValue.setEnabled(false);
            jComboBoxSingleValueType.setEnabled(false);
            jButtonSendSingleValue.setEnabled(false);
        }

    }//GEN-LAST:event_jRadioButtonSingleItemStateChanged

    private void jRadioButtonSequenceItemStateChanged(java.awt.event.ItemEvent evt)
         {//GEN-FIRST:event_jRadioButtonSequenceItemStateChanged

        if(jRadioButtonSequence.isSelected()) {
            jSpinnerPeriod.setEnabled(true);
            jButtonStartStopSequence.setEnabled(true);
        } else {
            jSpinnerPeriod.setEnabled(false);
            jButtonStartStopSequence.setEnabled(false);
        }

    }//GEN-LAST:event_jRadioButtonSequenceItemStateChanged

    private void jButtonSendSingleValueActionPerformed(java.awt.event.ActionEvent
        evt) {//GEN-FIRST:event_jButtonSendSingleValueActionPerformed
```

```
        if(jComboBoxSingleValueType.getSelectedIndex() == 0) {
            try {
                int value = Integer.valueOf(jTextFieldSingleValue.getText());
                sendValue(value);
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(this,
                    "Not a number.",
                    "Error",
                    JOptionPane.ERROR_MESSAGE);
            }
        } else {
            sendValue(jTextFieldSingleValue.getText());
        }

    }//GEN-LAST:event_jButtonSendSingleValueActionPerformed

    private void jButtonExitActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
        FIRST:event_jButtonExitActionPerformed

        if(ctx != null) {
            try {
                ctx.exit();
            } catch (OperationNotAllowedException ex) {
            jTextFieldStatus.setText(ex.toString());
                System.err.println(ex);
            }
        }

        this.setVisible(false);

    }//GEN-LAST:event_jButtonExitActionPerformed

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.ButtonGroup buttonGroup;
    private javax.swing.JButton jButtonExit;
    private javax.swing.JButton jButtonSendSingleValue;
    private javax.swing.JButton jButtonStartStopSequence;
    private javax.swing.JComboBox jComboBoxSingleValueType;
    private javax.swing.JLabel jLabelBuilding;
    private javax.swing.JLabel jLabelFloor;
    private javax.swing.JLabel jLabelPeriod;
    private javax.swing.JLabel jLabelRoom;
    private javax.swing.JLabel jLabelSecond;
    private javax.swing.JLabel jLabelSensorType;
    private javax.swing.JLabel jLabelValue;
    private javax.swing.JPanel jPanelSendValue;
    private javax.swing.JPanel jPanelSensorEmulator;
    private javax.swing.JPanel jPanelStatus;
    private javax.swing.JRadioButton jRadioButtonSequence;
    private javax.swing.JRadioButton jRadioButtonSingle;
    private javax.swing.JSpinner jSpinnerPeriod;
    private javax.swing.JTextField jTextFieldBuilding;
    private javax.swing.JTextField jTextFieldFloor;
    private javax.swing.JTextField jTextFieldRoom;
    private javax.swing.JTextField jTextFieldSensorType;
    private javax.swing.JTextField jTextFieldSingleValue;
    private javax.swing.JTextField jTextFieldStatus;
    // End of variables declaration//GEN-END:variables
}
```

Listing H.10 – JDialogEmulateSensor Class

Annexe I

package be.woine.thesis.model

```java
package be.woine.thesis.model;

import be.woine.thesis.building.common.Building;
import be.woine.thesis.building.common.Floor;
import be.woine.thesis.building.common.Room;
import be.woine.thesis.building.common.Sensor;
import java.util.ArrayList;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.TreeModel;
import javax.swing.tree.TreePath;

/**
 *
 * @author Remy
 * @version 1.0
 */
public class BuildingTreeModel implements TreeModel {

    private Building building;
    private ArrayList<TreeModelListener> treeModelListeners = new ArrayList<
        TreeModelListener>();

    public BuildingTreeModel() {
        this.building = new Building();
    }

    public BuildingTreeModel(Building building) {
        this.building = building;
    }

    public void reload() {
        fireTreeStructureChanged(building);
    }

    protected void fireTreeStructureChanged(Building building) {
        TreeModelEvent e = new TreeModelEvent(this, new Object[] {building});
        for (TreeModelListener tml : treeModelListeners) {
            tml.treeStructureChanged(e);
        }
    }

    public Building getBuilding() {
        return this.building;
    }

    public void addFloor(Floor floor) {
        building.addFloor(floor);
        fireTreeStructureChanged(building);
    }
```

```
    public void removeFloor(int floorIndex) {
        building.getFloorList().remove(floorIndex);
        fireTreeStructureChanged(building);
    }

    public void addRoom(int floorIndex, Room room) {
        building.getFloor(floorIndex).addRoom(room);
        fireTreeStructureChanged(building);
    }

    public void removeRoom(int floorIndex, int roomIndex) {
        building.getFloor(floorIndex).getRoomList().remove(roomIndex);
        fireTreeStructureChanged(building);
    }

    public void addSensor(int floorIndex, int roomIndex, Sensor sensor) {
        building.getFloor(floorIndex).getRoom(roomIndex).addSensor(sensor);
        //fireTreeStructureChanged(building);
    }

    public void removeSensor(int floorIndex, int roomIndex, int sensorIndex) {
        building.getFloor(floorIndex).getRoom(roomIndex).getSensorList().remove(
            sensorIndex);
    }

    public ArrayList<Sensor> getSensorList(int floorIndex, int roomIndex) {
        return building.getFloor(floorIndex).getRoom(roomIndex).getSensorList();
    }

    @Override
    public Object getRoot() {
        return this.building;
    }

    @Override
    public Object getChild(Object parent, int index) {

        if(parent.getClass() == Building.class)
            return ((Building)parent).getFloor(index);
        else
            return ((Floor)parent).getRoom(index);

    }

    @Override
    public int getChildCount(Object parent) {

        if(parent.getClass() == Building.class)
            return ((Building)parent).getFloorCount();
        else
            return ((Floor)parent).getRoomCount();

    }

    @Override
    public boolean isLeaf(Object node) {

        if(node.getClass() == Room.class)
            return true;
        else
            return false;

    }

    @Override
    public void valueForPathChanged(TreePath path, Object newValue) {
        System.out.println("*** valueForPathChanged : " + path + " --> " + newValue
            );
    }
```

```
    @Override
    public int getIndexOfChild(Object parent, Object child) {

        if((parent == null) || (child == null))
            return -1;

        if(parent.getClass() == Building.class)
            return ((Building)parent).getFloorList().indexOf((Floor)child);
        else
            return ((Floor)parent).getRoomList().indexOf((Room)child);

    }

    @Override
    public void addTreeModelListener(TreeModelListener l) {
        treeModelListeners.add(l);
    }

    @Override
    public void removeTreeModelListener(TreeModelListener l) {
        treeModelListeners.remove(l);
    }

}
```

Listing I.1 – BuildingTreeModel Class

```
package be.woine.thesis.model;

import be.woine.thesis.rule.common.SimpleRule;
import java.util.ArrayList;
import javax.swing.table.AbstractTableModel;

/**
 *
 * @author Remy
 * @version 1.0
 */
public class CompoundRuleAbstractTableModel extends AbstractTableModel {

    private ArrayList<SimpleRule> ruleList;
    private String[] title = {"Type", "Rule"};

    public CompoundRuleAbstractTableModel() {
        this.ruleList = new ArrayList<SimpleRule>();
    }

    public CompoundRuleAbstractTableModel(ArrayList<SimpleRule> ruleList) {
        this.ruleList = ruleList;
    }

    public void addRule(SimpleRule rule) {
        this.ruleList.add(rule);
        this.fireTableStructureChanged();
    }

    public void removeRule(int index) {
        this.ruleList.remove(index);
        this.fireTableStructureChanged();
    }

    public ArrayList<SimpleRule> getRuleList() {
        return this.ruleList;
    }

    @Override
    public int getRowCount() {
        return this.ruleList.size();
    }
```

```
    @Override
    public int getColumnCount() {
        return this.title.length;
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        if(columnIndex == 0) {
            return this.ruleList.get(rowIndex).getType();
        } else {
            return this.ruleList.get(rowIndex);
        }
    }

    @Override
    public String getColumnName(int column) {
        return this.title[column];
    }

}
```

Listing I.2 – CompoundRuleAbstractTableModel Class

```
package be.woine.thesis.model;

import be.woine.thesis.reaction.common.Reaction;
import java.util.ArrayList;
import javax.swing.table.AbstractTableModel;

/**
 *
 * @author Remy
 * @version 1.0
 */
public class ReactionAbstractTableModel extends AbstractTableModel {

    private ArrayList<Reaction> reactionList;
    private String[] title = {"Settled", "Event", "Body"};

    public ReactionAbstractTableModel() {
        this.reactionList = new ArrayList<Reaction>();
    }

    public ReactionAbstractTableModel(ArrayList<Reaction> reactionList) {
        this.reactionList = reactionList;
    }

    public ArrayList<Reaction> getReactionList() {
        return this.reactionList;
    }

    @Override
    public int getRowCount() {
        return reactionList.size();
    }

    @Override
    public int getColumnCount() {
        return this.title.length;
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        if(columnIndex == 0)
            return this.reactionList.get(rowIndex).isSettled();
        else if(columnIndex == 1)
            return this.reactionList.get(rowIndex).getEvent();
        else
```

```
            return this.reactionList.get(rowIndex).getBody();
    }

    @Override
    public String getColumnName(int column) {
        return this.title[column];
    }

    @Override
    public Class getColumnClass(int columnIndex) {
        if(columnIndex == 0)
            return this.reactionList.get(0).isSettled().getClass();
        else if(columnIndex == 1)
            return this.reactionList.get(0).getEvent().getClass();
        else
            return this.reactionList.get(0).getBody().getClass();
    }

}
```

Listing I.3 – ReactionAbstractTableModel Class

```
package be.woine.thesis.model;

import be.woine.thesis.rule.common.Rule;
import java.util.ArrayList;
import javax.swing.table.AbstractTableModel;

/**
 *
 * @author Remy
 * @version 1.0
 */
public class RuleAbstractTableModel extends AbstractTableModel {

    private ArrayList<Rule> ruleList;
    private String[] title = {"#", "Type", "Rule", "Action"};

    public RuleAbstractTableModel() {
        this.ruleList = new ArrayList<Rule>();
    }

    public RuleAbstractTableModel(ArrayList<Rule> ruleList) {
        this.ruleList = ruleList;
    }

    public void addRule(Rule rule) {
        this.ruleList.add(rule);
        this.fireTableStructureChanged();
    }

    public void removeRule(int index) {
        this.ruleList.remove(index);
        this.fireTableStructureChanged();
    }

    public ArrayList<Rule> getRuleList() {
        return this.ruleList;
    }

    @Override
    public int getRowCount() {
        return this.ruleList.size();
    }

    @Override
    public int getColumnCount() {
        return this.title.length;
    }
```

```
    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        if(columnIndex == 0) {
            return this.ruleList.get(rowIndex).getId();
        } else if(columnIndex == 1) {
            return this.ruleList.get(rowIndex).getType();
        } else if(columnIndex == 2) {
            return this.ruleList.get(rowIndex);
        } else {
            return this.ruleList.get(rowIndex).getAction();
        }
    }

    @Override
    public String getColumnName(int column) {
        return this.title[column];
    }

}
```

Listing I.4 – RuleAbstractTableModel Class

```
package be.woine.thesis.model;

import be.woine.thesis.rule.common.CompoundRule;
import java.util.ArrayList;
import javax.swing.table.AbstractTableModel;

/**
 *
 * @author Remy
 * @version 1.0
 */
public class SequenceRuleAbstractTableModel extends AbstractTableModel {

    private ArrayList<CompoundRule> ruleList;
    private String[] title = {"Type", "Rule"};

    public SequenceRuleAbstractTableModel() {
        this.ruleList = new ArrayList<CompoundRule>();
    }

    public SequenceRuleAbstractTableModel(ArrayList<CompoundRule> ruleList) {
        this.ruleList = ruleList;
    }

    public void addRule(CompoundRule rule) {
        this.ruleList.add(rule);
        this.fireTableStructureChanged();
    }

    public void removeRule(int index) {
        this.ruleList.remove(index);
        this.fireTableStructureChanged();
    }

    public ArrayList<CompoundRule> getRuleList() {
        return this.ruleList;
    }

    @Override
    public int getRowCount() {
        return this.ruleList.size();
    }

    @Override
    public int getColumnCount() {
        return this.title.length;
```

```
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        if(columnIndex == 0) {
            return this.ruleList.get(rowIndex).getType();
        } else {
            return this.ruleList.get(rowIndex);
        }
    }

    @Override
    public String getColumnName(int column) {
        return this.title[column];
    }

}
```

Listing I.5 – SequenceRuleAbstractTableModel Class

```
package be.woine.thesis.model;

import be.woine.thesis.node.common.NodeEnvironment;
import be.woine.thesis.node.common.TupleCentre;
import java.util.ArrayList;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.tree.TreeModel;
import javax.swing.tree.TreePath;

/**
 *
 * @author Remy
 * @version 1.0
 */
public class TupleCentreTreeModel implements TreeModel {

    private NodeEnvironment node;
    private ArrayList<TreeModelListener> treeModelListeners = new ArrayList<
        TreeModelListener>();

    public TupleCentreTreeModel() {
    }

    public TupleCentreTreeModel(NodeEnvironment node) {
        this.node = node;
    }

    public void reload() {
        fireTreeStructureChanged(node);
    }

    public NodeEnvironment getNode() {
        return this.node;
    }

    protected void fireTreeStructureChanged(NodeEnvironment node) {
        TreeModelEvent e = new TreeModelEvent(this, new Object[] {node});
        for (TreeModelListener tml : treeModelListeners) {
            tml.treeStructureChanged(e);
        }
    }

    @Override
    public Object getRoot() {
        return this.node;
    }

    @Override
```

```
    public Object getChild(Object parent, int index) {
        return ((NodeEnvironment)parent).getTupleCentre(index);
    }

    @Override
    public int getChildCount(Object parent) {
        return ((NodeEnvironment)parent).getTupleCentreCount();
    }

    @Override
    public boolean isLeaf(Object node) {
        if(node.getClass() == TupleCentre.class)
            return true;
        else
            return false;
    }

    @Override
    public void valueForPathChanged(TreePath path, Object newValue) {
        System.out.println("*** valueForPathChanged : " + path + " --> " + newValue
            );
    }

    @Override
    public int getIndexOfChild(Object parent, Object child) {
        if((parent == null) || (child == null))
            return -1;

        if(parent.getClass() == NodeEnvironment.class)
            return ((NodeEnvironment)parent).getTupleCentreList().indexOf((
                TupleCentre)child);
        else
            return -1;
    }

    @Override
    public void addTreeModelListener(TreeModelListener l) {
        treeModelListeners.add(l);
    }

    @Override
    public void removeTreeModelListener(TreeModelListener l) {
        treeModelListeners.remove(l);
    }

}
```

Listing I.6 – TupleCentreTreeModel Class

```
package be.woine.thesis.model;

import be.woine.thesis.rule.common.ValueSource;
import java.util.ArrayList;
import javax.swing.table.AbstractTableModel;

/**
 *
 * @author Remy
 * @version 1.0
 */
public class ValueSourceAbstractTableModel extends AbstractTableModel {

    private ArrayList<ValueSource> sensorList;
    private String[] title = {"Type", "ID"};

    public ValueSourceAbstractTableModel() {
        this.sensorList = new ArrayList<ValueSource>();
    }
```

```
    public ValueSourceAbstractTableModel(ArrayList<ValueSource> sensorList) {
        this.sensorList = sensorList;
    }

    public void reload() {
        this.fireTableStructureChanged();
    }

    public void addValueSource(ValueSource valueSource) {
        this.sensorList.add(valueSource);
        this.fireTableStructureChanged();
    }

    public void removeValueSource(int index) {
        this.sensorList.remove(index);
        this.fireTableStructureChanged();
    }

    @Override
    public int getRowCount() {
        return this.sensorList.size();
    }

    @Override
    public int getColumnCount() {
        return this.title.length;
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        ValueSource source = this.sensorList.get(rowIndex);
        if(columnIndex == 0) {
            return source.building.getFloor(source.floorIndex).getRoom(source.
                roomIndex).getSensor(source.sensorIndex).getType();
        } else {
            return source.building.getFloor(source.floorIndex).getRoom(source.
                roomIndex).getSensor(source.sensorIndex).getId();
        }
    }

    @Override
    public String getColumnName(int column) {
        return this.title[column];
    }

    public ValueSource getElementAt(int rowIndex) {
        return this.sensorList.get(rowIndex);
    }

}
```

Listing I.7 – ValueSourceAbstractTableModel Class

Annexe J

package be.woine.thesis.utilz

```java
package be.woine.thesis.utilz;

import alice.tucson.api.TucsonException;
import alice.tucson.service.Node;

/**
 * This class intends to create a thread handling a TuCSoN Node.
 * @author Remy
 * @version 1.0
 */
public class LaunchNode extends Thread {

    private Node node;

    /**
     * Constructor of the LaunchNode class.
     * @throws TucsonException Exception thrown by the TuCSoN Node.
     */
    public LaunchNode() throws TucsonException {
        node = new Node();
    }

    /**
     * Override of the run() method. This involves the node installation.
     */
    @Override
    public void run() {
        node.install();
    }

}
```

Listing J.1 – LaunchNode Class

```java
package be.woine.thesis.utilz;

import be.woine.thesis.node.common.TupleCentre;
import be.woine.thesis.reaction.common.Reaction;
import be.woine.thesis.rule.common.Rule;
import be.woine.thesis.rule.common.RuleOperator;
import be.woine.thesis.rule.common.RuleType;
import be.woine.thesis.rule.common.SimpleRule;
import be.woine.thesis.rule.common.ValueSource;
import be.woine.thesis.utilz.exception.InvalidOperatorException;
import be.woine.thesis.utilz.exception.UndefinedRuleException;
import java.util.ArrayList;
import java.util.Collections;

/**
 *
```

```
 * @author Remy
 * @version 1.0
 */
public class UniversalParser {

    public static ArrayList<Reaction> getReactionFromRule(Rule rule, int
        samplingTime) throws UndefinedRuleException, InvalidOperatorException {

        ArrayList<Reaction> reactionList = new ArrayList<Reaction>();

        /* Get Reaction from Fixed Simple Rule */
        if(rule.getType() == RuleType.FIXED) {

            // Get TupleCentreId
            ValueSource source = rule.getCompoundRule(0).getSimpleRule(0).getSource
                ();
            String tupleCentreId = source.building.getFloor(source.floorIndex).
                getName() + "_" + source.building.getFloor(source.floorIndex).
                getRoom(source.roomIndex).getName();

            // Get Event and Body
            ArrayList<Reaction> innerReactionList = getReactionFromSimpleRule(rule.
                getCompoundRule(0).getSimpleRule(0));
            ValueSource innerSource = rule.getCompoundRule(0).getSimpleRule(0).
                getSource();

            // Set Action in Body
            //String action = "out_tc(action_centre,action(" + rule.getId() + "))";
            String action = "out_tc(action_centre,action(" + rule.getId() + ",ids([
                Id]),values([" + innerSource.building.getFloor(innerSource.
                floorIndex).getRoom(innerSource.roomIndex).getSensor(innerSource.
                sensorIndex).getType().getVar() + "])" + "))";

            innerReactionList.get(0).appendBody(action);
            innerReactionList.get(0).setTupleCentreId(tupleCentreId);

            // Add to main reaction list
            reactionList.addAll(innerReactionList);

        /* Get Reaction from Range Simple Rule */
        } else if(rule.getType() == RuleType.RANGE) {

            // Get TupleCentreId
            ValueSource source = rule.getCompoundRule(0).getSimpleRule(0).getSource
                ();
            String tupleCentreId = source.building.getFloor(source.floorIndex).
                getName() + "_" + source.building.getFloor(source.floorIndex).
                getRoom(source.roomIndex).getName();

            // Get Event and Body
            ArrayList<Reaction> innerReactionList = getReactionFromSimpleRule(rule.
                getCompoundRule(0).getSimpleRule(0));
            ValueSource innerSource = rule.getCompoundRule(0).getSimpleRule(0).
                getSource();

            // Set Action in Body
            //String action = "out_tc(action_centre,action(" + rule.getId() + "))";
            String action = "out_tc(action_centre,action(" + rule.getId() + ",ids([
                Id]),values([" + innerSource.building.getFloor(innerSource.
                floorIndex).getRoom(innerSource.roomIndex).getSensor(innerSource.
                sensorIndex).getType().getVar() + "])" + "))";

            // In the case of the SpecialOperator is NOT_INCLUDED_IN, two reactions
                are generated for the same tuple centre
            // In the case of the SpecialOperator is INCLUDED_IN, one reaction with
                two conditions is generated in the same tuple centre
            for(int i = 0; i < innerReactionList.size(); i++) {
                innerReactionList.get(i).appendBody(action);
                innerReactionList.get(i).setTupleCentreId(tupleCentreId);
            }
```

```
            // Add to main reaction list
            reactionList.addAll(innerReactionList);

        /* Get Reaction from Compound Rule */
        } else if(rule.getType() == RuleType.COMPOUND) {

            int compoundId = rule.getCompoundRule(0).getId();
            int term = 0;

            for(int i = 0; i < rule.getCompoundRule(0).getSimpleRuleList().size();
                i++) {

                ArrayList<Reaction> innerReactionList = getReactionFromSimpleRule(
                    rule.getCompoundRule(0).getSimpleRule(i));

                for(int a = 0; a < innerReactionList.size(); a++) {

                    // Get TupleCentreId
                    ValueSource innerSource = rule.getCompoundRule(0).getSimpleRule
                        (i).getSource();
                    String tupleCentreId = innerSource.building.getFloor(
                        innerSource.floorIndex).getName() + "_" + innerSource.
                        building.getFloor(innerSource.floorIndex).getRoom(
                        innerSource.roomIndex).getName();

                    // Set Compound relative action
                    innerReactionList.get(a).appendBody("out_tc(compound_" +
                        compoundId + ",term(" + term + "),Id," + innerSource.
                        building.getFloor(innerSource.floorIndex).getRoom(
                        innerSource.roomIndex).getSensor(innerSource.sensorIndex).
                        getType().getVar() + "))");
                    innerReactionList.get(a).setTupleCentreId(tupleCentreId);

                    // Add reaction for each terms of the compound rule
                    reactionList.add(innerReactionList.get(a));

                    // Increment the number of terms of the CompoundRule
                    term++;

                }

            }

            // Insert a first tuple all_term(-1,-1,..,-1) at the first time and
                initialize list
            Reaction firstAllTermReaction = new Reaction();
            firstAllTermReaction.setEvent("out(term(X,_,_))");
            firstAllTermReaction.setBody("no_r(" + getAllTermPredicate("all_term",
                term, term + 1, true) + ")");
            firstAllTermReaction.appendBody("out_r(" + getAllTermPredicate("
                all_term", term, -1, false) + ")");
            firstAllTermReaction.appendBody("out_r(" + getAllTermPredicate("all_id"
                , term, -1, false) + ")");
            firstAllTermReaction.appendBody("out_r(" + getAllTermPredicate("
                all_value", term, -1, false) + ")");
            firstAllTermReaction.setTupleCentreId("compound_" + compoundId);

            reactionList.add(firstAllTermReaction);

            // Add all reactions managing all the terms of the compound rule as a
                whole
            reactionList.addAll(generateTermOfCompound(term, "compound_" +
                compoundId));

            // Manage the sampling time
            int croppedSamplingTime = (int)(samplingTime * 0.95);

            Reaction timedReaction0 = new Reaction();
            timedReaction0.setEvent("out(term(X,Y,Z))");
```

```
            timedReaction0.setBody("no_r(first)");
            timedReaction0.appendBody("out_r(first)");
            timedReaction0.appendBody("new_trap(Id," + croppedSamplingTime + ",
                lease_expired(term(X,Y,Z)))");
            timedReaction0.appendBody("out_r(trap_id(Id))");
            timedReaction0.setTupleCentreId("compound_" + compoundId);

            Reaction timedReaction1 = new Reaction();
            timedReaction1.setEvent("trap(lease_expired(term(X,Y,Z)))");
            timedReaction1.setBody("in_r(" + getAllTermPredicate("all_term", term,
                term + 1, true) + ")");
            timedReaction1.appendBody("in_r(" + getAllTermPredicate("all_id", term,
                 term + 1, true) + ")");
            timedReaction1.appendBody("in_r(" + getAllTermPredicate("all_value",
                term, term + 1, true) + ")");
            timedReaction1.appendBody("out_r(" + getAllTermPredicate("all_term",
                term, -1, false) + ")");
            timedReaction1.appendBody("out_r(" + getAllTermPredicate("all_id", term
                , -1, false) + ")");
            timedReaction1.appendBody("out_r(" + getAllTermPredicate("all_value",
                term, -1, false) + ")");
            timedReaction1.appendBody("in_r(first)");
            timedReaction1.appendBody("in_r(trap_id(Id))");
            timedReaction1.setTupleCentreId("compound_" + compoundId);

            reactionList.add(timedReaction0);
            reactionList.add(timedReaction1);


            // Add reaction to deal with the action
            Reaction actionReaction = new Reaction();
            actionReaction.setEvent("out_r(" + getAllTermPredicate("all_term", term
                , term, false) + ")");
            actionReaction.setBody("in_r(" + getAllTermPredicate("all_term", term,
                term, false) + ")");
            actionReaction.appendBody("in_r(" + getAllTermPredicate("all_id", term,
                 term + 1, false) + ")");
            actionReaction.appendBody("IdList=[" + getAllTerm(term, "I") + "]");
            actionReaction.appendBody("in_r(" + getAllTermPredicate("all_value",
                term, term + 1, false) + ")");
            actionReaction.appendBody("ValueList=[" + getAllTerm(term, "V") + "]");
            actionReaction.appendBody("out_r(" + getAllTermPredicate("all_term",
                term, -1, false) + ")");
            actionReaction.appendBody("out_r(" + getAllTermPredicate("all_id", term
                , -1, false) + ")");
            actionReaction.appendBody("out_r(" + getAllTermPredicate("all_value",
                term, -1, false) + ")");
            actionReaction.appendBody("rd_r(trap_id(Id))");
            actionReaction.appendBody("kill_trap(Id)");
            actionReaction.appendBody("in_r(first)");
            actionReaction.appendBody("in_r(trap_id(Id))");
            actionReaction.appendBody("out_tc(action_centre,action(" + rule.getId()
                 + ",ids(IdList),values(ValueList)))");
            actionReaction.setTupleCentreId("compound_" + compoundId);

            reactionList.add(actionReaction);


        /* Get Reaction from Sequence Rule */
        } else if(rule.getType() == RuleType.SEQUENCE) {



        } else {
            throw new UndefinedRuleException();
        }


        return reactionList;
    }
```

```
    private static ArrayList<Reaction> getReactionFromSimpleRule(SimpleRule rule)
        throws UndefinedRuleException, InvalidOperatorException {

        ArrayList<Reaction> reactionList = new ArrayList<Reaction>();

        /* Event : out(type(Id,Var)) */
        String event = "out(";
        event += rule.getSource().building.getFloor(rule.getSource().floorIndex).
            getRoom(rule.getSource().roomIndex).getSensor(rule.getSource().
            sensorIndex).getType();
        event += "(";
        //event += rule.getSource().building.getFloor(rule.getSource().floorIndex).
            getRoom(rule.getSource().roomIndex).getSensor(rule.getSource().
            sensorIndex).getId();
        event += "Id";
        event += ",";
        event += rule.getSource().building.getFloor(rule.getSource().floorIndex).
            getRoom(rule.getSource().roomIndex).getSensor(rule.getSource().
            sensorIndex).getType().getVar();
        event += ")";
        event += ")";

        if(rule.getType() == RuleType.FIXED) {

            Reaction reaction = new Reaction();

            /* Body : Var<op>Value */
            String body = rule.getSource().building.getFloor(rule.getSource().
                floorIndex).getRoom(rule.getSource().roomIndex).getSensor(rule.
                getSource().sensorIndex).getType().getVar();
            body += rule.getRuleBody(0).getOperator().getOperator();
            body += rule.getRuleBody(0).getValue();


            reaction.setEvent(event);
            reaction.setBody(body);

            reactionList.add(reaction);

        } else if(rule.getType() == RuleType.RANGE) {

            if(rule.getSpecialOperator() == RuleOperator.INCLUDED_IN) {

                Reaction reaction = new Reaction();

                /* Body : Var<op0>Value0,Var<op1>Value1 */
                String body0 = rule.getSource().building.getFloor(rule.getSource().
                    floorIndex).getRoom(rule.getSource().roomIndex).getSensor(rule.
                    getSource().sensorIndex).getType().getVar();
                body0 += rule.getRuleBody(0).getOperator().getOperator();
                body0 += rule.getRuleBody(0).getValue();

                String body1 = rule.getSource().building.getFloor(rule.getSource().
                    floorIndex).getRoom(rule.getSource().roomIndex).getSensor(rule.
                    getSource().sensorIndex).getType().getVar();
                body1 += rule.getRuleBody(1).getOperator().getOperator();
                body1 += rule.getRuleBody(1).getValue();


                reaction.setEvent(event);
                reaction.appendBody(body0);
                reaction.appendBody(body1);

                reactionList.add(reaction);

            } else if(rule.getSpecialOperator() == RuleOperator.NOT_INCLUDED_IN) {

                Reaction reaction0 = new Reaction();
                Reaction reaction1 = new Reaction();
```

```
                /* Body0 : Var<!op0>Value0 */
                String body0 = rule.getSource().building.getFloor(rule.getSource().
                    floorIndex).getRoom(rule.getSource().roomIndex).getSensor(rule.
                    getSource().sensorIndex).getType().getVar();

                if(rule.getRuleBody(0).getOperator() == RuleOperator.GREATER_THAN)
                    {
                    body0 += RuleOperator.SMALLER_THAN_OR_EQUAL_TO.getOperator();
                } else if(rule.getRuleBody(0).getOperator() == RuleOperator.
                    GREATER_THAN_OR_EQUAL_TO) {
                    body0 += RuleOperator.SMALLER_THAN.getOperator();
                } else if(rule.getRuleBody(0).getOperator() == RuleOperator.
                    SMALLER_THAN) {
                    body0 += RuleOperator.GREATER_THAN_OR_EQUAL_TO.getOperator();
                } else if(rule.getRuleBody(0).getOperator() == RuleOperator.
                    SMALLER_THAN_OR_EQUAL_TO) {
                    body0 += RuleOperator.GREATER_THAN.getOperator();
                } else {
                    throw new InvalidOperatorException();
                }

                body0 += rule.getRuleBody(0).getValue();

                /* Body1 : Var<!op1>Value1 */
                String body1 = rule.getSource().building.getFloor(rule.getSource().
                    floorIndex).getRoom(rule.getSource().roomIndex).getSensor(rule.
                    getSource().sensorIndex).getType().getVar();

                if(rule.getRuleBody(1).getOperator() == RuleOperator.GREATER_THAN)
                    {
                    body1 += RuleOperator.SMALLER_THAN_OR_EQUAL_TO.getOperator();
                } else if(rule.getRuleBody(1).getOperator() == RuleOperator.
                    GREATER_THAN_OR_EQUAL_TO) {
                    body1 += RuleOperator.SMALLER_THAN.getOperator();
                } else if(rule.getRuleBody(1).getOperator() == RuleOperator.
                    SMALLER_THAN) {
                    body1 += RuleOperator.GREATER_THAN_OR_EQUAL_TO.getOperator();
                } else if(rule.getRuleBody(1).getOperator() == RuleOperator.
                    SMALLER_THAN_OR_EQUAL_TO) {
                    body1 += RuleOperator.GREATER_THAN.getOperator();
                } else {
                    throw new InvalidOperatorException();
                }

                body1 += rule.getRuleBody(1).getValue();


                reaction0.setEvent(event);
                reaction0.setBody(body0);
                reaction1.setEvent(event);
                reaction1.setBody(body1);

                reactionList.add(reaction0);
                reactionList.add(reaction1);

            } else {
                throw new InvalidOperatorException();
            }

        } else {
            throw new UndefinedRuleException();
        }

        return reactionList;

    }


    private static ArrayList<Reaction> generateTermOfCompound(int numberOfTerms,
```

```
        String tupleCentreId) {

        ArrayList<Reaction> reactionList = new ArrayList<Reaction>();

        Reaction reaction;

        for(int i = 0; i < numberOfTerms; i++) {

            reaction = new Reaction();

            reaction.setEvent("out(term(" + i + ",Sid,Value))");

            reaction.setBody("in_r(term(" + i + ",Sid,Value))");
            reaction.appendBody("in_r(" + getAllTermPredicate("all_term",
                numberOfTerms, numberOfTerms + 1, false) + ")");
            reaction.appendBody("out_r(" + getAllTermPredicate("all_term",
                numberOfTerms, i, false) + ")");
            reaction.appendBody("in_r(" + getAllTermPredicate("all_id",
                numberOfTerms, numberOfTerms + 1, false) + ")");
            reaction.appendBody("out_r(" + getAllTermPredicate("all_id",
                numberOfTerms, i, false, "Sid") + ")");
            reaction.appendBody("in_r(" + getAllTermPredicate("all_value",
                numberOfTerms, numberOfTerms + 1, false) + ")");
            reaction.appendBody("out_r(" + getAllTermPredicate("all_value",
                numberOfTerms, i, false, "Value") + ")");

            reaction.setTupleCentreId(tupleCentreId);

            reactionList.add(reaction);
        }

        return reactionList;

    }


    private static String getAllTermPredicate(String atom, int numberOfTerms, int
        indexOfTerm, boolean anonyme) {

        String predicate = atom + "(";
        String var = "";

        if(atom.compareTo("all_id") == 0)
            var = "I";
        else if(atom.compareTo("all_value") == 0)
            var = "V";
        else
            var = "X";

        for(int i = 0; i < numberOfTerms; i++) {

            if(anonyme) {
                predicate += "_";
            } else {
                if(indexOfTerm == -1) {
                    predicate += "-1";
                } else if(indexOfTerm == numberOfTerms) {
                    predicate += i;
                } else {
                    if(i == indexOfTerm) {
                        predicate += indexOfTerm;
                    } else {
                        predicate += (var + i);
                    }
                }
            }

            if(i != numberOfTerms - 1)
                predicate += ",";
        }
```

```
        predicate += ")";

        return predicate;

    }

    private static String getAllTermPredicate(String atom, int numberOfTerms, int
        indexOfTerm, boolean anonyme, String termName) {

        String predicate = atom + "(";
        String var = "";

        if(atom.compareTo("all_id") == 0)
            var = "I";
        else if(atom.compareTo("all_value") == 0)
            var = "V";
        else
            var = "X";

        for(int i = 0; i < numberOfTerms; i++) {

            if(anonyme) {
                predicate += "_";
            } else {
                if(indexOfTerm == -1) {
                    predicate += "-1";
                } else if(indexOfTerm == numberOfTerms) {
                    predicate += i;
                } else {
                    if(i == indexOfTerm) {
                        predicate += termName;
                    } else {
                        predicate += (var + i);
                    }
                }
            }

            if(i != numberOfTerms - 1)
                predicate += ",";
        }

        predicate += ")";

        return predicate;

    }

    private static String getAllTerm(int numberOfTerms, String varName) {

        String predicate = "";

        for(int i = 0; i < numberOfTerms; i++) {
            predicate += (varName + i);
            if(i != numberOfTerms - 1)
                predicate += ",";
        }

        return predicate;

    }

    public static ArrayList<TupleCentre> getTupleCentreListFromReaction(ArrayList<
        Reaction> reactionList) {

        ArrayList<TupleCentre> tupleCentreList = new ArrayList<TupleCentre>();

        for(int i = 0; i < reactionList.size(); i++) {

            if(!tupleCentreList.isEmpty()) {
```

```
                int value = getTupleCentreIndexFromId(tupleCentreList, reactionList
                    .get(i).getTupleCentreId());

                if(value == -1) {

                    // No TupleCentre with this ID --> Create it & Add reaction
                    tupleCentreList.add(new TupleCentre(reactionList.get(i).
                        getTupleCentreId()));
                    tupleCentreList.get(getTupleCentreIndexFromId(tupleCentreList,
                        reactionList.get(i).getTupleCentreId())).addReaction(
                        reactionList.get(i));

                } else {

                    // TupleCentre already created --> Add reaction
                    tupleCentreList.get(value).addReaction(reactionList.get(i));

                }

            } else {

                // Add a first TupleCentre with its reaction
                tupleCentreList.add(new TupleCentre(reactionList.get(i).
                    getTupleCentreId()));
                tupleCentreList.get(getTupleCentreIndexFromId(tupleCentreList,
                    reactionList.get(i).getTupleCentreId())).addReaction(
                    reactionList.get(i));

            }
        }

        // Sort by alphabetic order
        Collections.sort(tupleCentreList);

        return tupleCentreList;
    }

    private static int getTupleCentreIndexFromId(ArrayList<TupleCentre>
        tupleCentreList, String id) {

        int value = -1;

        for(int i = 0; i < tupleCentreList.size(); i++) {
            if(tupleCentreList.get(i).getId().compareTo(id) == 0) {
                value = i;
                break;
            }
        }

        return value;

    }

    public static Reaction getMonitorReaction(String tupleCentreId) {

        Reaction reaction = new Reaction();

        reaction.setEvent("out(X)");
        reaction.setBody("current_tc(N)");
        reaction.appendBody("current_tuple(T)");
        reaction.appendBody("out_tc(monitor_centre,monitor(N,T))");
        reaction.setTupleCentreId(tupleCentreId);

        return reaction;
    }

    public static ArrayList<Reaction> getLeaseTimeReaction(String tupleCentreId,
        int samplingTime) {
```

```
        ArrayList<Reaction> reactionList = new ArrayList<Reaction>();

        reactionList.add(new Reaction());
        reactionList.add(new Reaction());

        reactionList.get(0).setEvent("out(T)");
        reactionList.get(0).setBody("new_trap(Id," + samplingTime + ",lease_expired
            (T))");
        reactionList.get(0).setTupleCentreId(tupleCentreId);

        reactionList.get(1).setEvent("trap(lease_expired(T))");
        reactionList.get(1).setBody("in_r(T)");
        reactionList.get(1).setTupleCentreId(tupleCentreId);

        return reactionList;
    }

}
```

Listing J.2 – UniversalParser Class

Annexe K

package be.woine.thesis.utilz.exception

```java
package be.woine.thesis.utilz.exception;

/**
 *
 * @author Remy
 * @version 1.0
 */
public class InvalidOperatorException extends UniversalParserException {

}
```

Listing K.1 – InvalidOperatorException Class

```java
package be.woine.thesis.utilz.exception;

/**
 *
 * @author Remy
 * @version 1.0
 */
public class UndefinedRuleException extends UniversalParserException {

}
```

Listing K.2 – UndefinedRuleException Class

```java
package be.woine.thesis.utilz.exception;

/**
 * Generic UniversalParser exception.
 * @author Remy
 * @version 1.0
 */
public class UniversalParserException extends Exception {

    /**
     * Constructor of the UniversalParserException object.
     * @param message A string giving explanation about the exception.
     */
    public UniversalParserException(String message) {
        super(message);
    }

    /**
     * Empty constructor of the UniversalParserException object.
```

```
     */
    public UniversalParserException() {
    }

}
```

Listing K.3 – UniversalParserException Class

Annexe L

package be.woine.thesis.environmentManagerImpl

```
package be.woine.thesis.environmentManagerImpl;

import be.woine.thesis.environmentManagerGui.EnvironmentManagerForm;

/**
 *
 * @author Remy
 * @version 1.0
 */
public class EnvironmentManager {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        EnvironmentManagerForm environmentManagerForm = new EnvironmentManagerForm
            ();
        environmentManagerForm.setVisible(true);

    }
}
```

Listing L.1 – EnvironmentManager Class

Annexe M

package be.woine.thesis.environmentManagerGui

```
/*
 * EnvironmentManagerForm.java
 *
 * Created on 15-mai-2012, 20:39:28
 */
package be.woine.thesis.environmentManagerGui;

import alice.logictuple.LogicTuple;
import alice.logictuple.Var;
import alice.tucson.api.AgentId;
import alice.tucson.api.ContextNotAvailableException;
import alice.tucson.api.InvalidSpecificationException;
import alice.tucson.api.InvalidTupleCentreIdException;
import alice.tucson.api.OperationNotAllowedException;
import alice.tucson.api.Tucson;
import alice.tucson.api.TucsonContextSynch;
import alice.tucson.api.TucsonException;
import alice.tucson.api.TupleCentreId;
import alice.tucson.api.UnreachableNodeException;
import be.woine.thesis.model.RuleAbstractTableModel;
import be.woine.thesis.model.BuildingTreeModel;
import be.woine.thesis.dialog.JDialogSimpleRuleFixed;
import be.woine.thesis.dialog.JDialogCompoundRule;
import be.woine.thesis.dialog.JDialogRuleChooser;
import be.woine.thesis.dialog.JDialogSimpleRuleRange;
import be.woine.thesis.action.common.Action;
import be.woine.thesis.action.common.ActionType;
import be.woine.thesis.agent.BuildingAgent;
import be.woine.thesis.agent.MonitorAgent;
import be.woine.thesis.agent.RulerAgent;
import be.woine.thesis.building.common.*;
import be.woine.thesis.dialog.JDialogSensor;
import be.woine.thesis.dialog.JDialogSequenceRule;
import be.woine.thesis.dialog.JDialogTreeElement;
import be.woine.thesis.model.ReactionAbstractTableModel;
import be.woine.thesis.model.TupleCentreTreeModel;
import be.woine.thesis.model.ValueSourceAbstractTableModel;
import be.woine.thesis.node.common.NodeEnvironment;
import be.woine.thesis.node.common.TupleCentre;
import be.woine.thesis.reaction.common.Reaction;
import be.woine.thesis.rule.common.CompoundRule;
import be.woine.thesis.rule.common.IntegerRuleBody;
import be.woine.thesis.rule.common.Rule;
import be.woine.thesis.rule.common.RuleOperator;
import be.woine.thesis.rule.common.RuleType;
import be.woine.thesis.rule.common.SimpleRule;
import be.woine.thesis.rule.common.ValueSource;
import be.woine.thesis.utilz.LaunchNode;
```

```
import be.woine.thesis.utilz.UniversalParser;
import be.woine.thesis.utilz.exception.InvalidOperatorException;
import be.woine.thesis.utilz.exception.UndefinedRuleException;
import java.util.ArrayList;
import javax.swing.JOptionPane;
import javax.swing.JTable;
import javax.swing.ListSelectionModel;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import javax.swing.tree.TreeSelectionModel;

/**
 *
 * @author Remy
 * @version 1.0
 */
public class EnvironmentManagerForm extends javax.swing.JFrame {

    private static final int JTABLE_SENSOR_COL0_WIDTH = 110;
    private static final int JTABLE_SENSOR_COL1_WIDTH = 46;

    private static final int JTABLE_RULE_COL0_WIDTH = 40;
    private static final int JTABLE_RULE_COL1_WIDTH = 80;
    private static final int JTABLE_RULE_COL2_WIDTH = 301;
    private static final int JTABLE_RULE_COL3_WIDTH = 80;

    private static final int JTABLE_REACTION_COL0_WIDTH = 50;
    private static final int JTABLE_REACTION_COL1_WIDTH = 146;
    private static final int JTABLE_REACTION_COL2_WIDTH = 2048;

    public Building building;
    public ArrayList<Rule> defaultRuleList;
    private LaunchNode launchNode = null;
    private RulerAgent rulerAgent = null;
    private BuildingAgent buildingAgent = null;
    private MonitorAgent monitorAgent = null;

    private TucsonContextSynch ctx;

    private boolean buttonPlay = true;
    private boolean nodeLaunched = false;

    /** Creates new form EnvironmentManagerForm */
    public EnvironmentManagerForm() {

        /* Look And Feel */
        try {
            javax.swing.UIManager.setLookAndFeel(javax.swing.UIManager.
                getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException ex) {
            System.out.println(ex.toString());
        } catch (InstantiationException ex) {
            System.out.println(ex.toString());
        } catch (IllegalAccessException ex) {
            System.out.println(ex.toString());
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            System.out.println(ex.toString());
        }

        initBuilding();
        initRule();
        initComponents();

        jButtonAddToTree.setEnabled(false);
        jButtonAddSensorToList.setEnabled(false);
        jButtonRemoveSensorFromList.setEnabled(false);

        jTableSensor.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        jTableSensor.getColumnModel().getColumn(0).setPreferredWidth(
            JTABLE_SENSOR_COL0_WIDTH);
```

```
        jTableSensor.getColumnModel().getColumn(1).setPreferredWidth(
            JTABLE_SENSOR_COL1_WIDTH);
        jTableSensor.getSelectionModel().setSelectionMode(ListSelectionModel.
            SINGLE_SELECTION);

        jTableRule.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        jTableRule.getColumnModel().getColumn(0).setPreferredWidth(
            JTABLE_RULE_COL0_WIDTH);
        jTableRule.getColumnModel().getColumn(1).setPreferredWidth(
            JTABLE_RULE_COL1_WIDTH);
        jTableRule.getColumnModel().getColumn(2).setPreferredWidth(
            JTABLE_RULE_COL2_WIDTH);
        jTableRule.getColumnModel().getColumn(3).setPreferredWidth(
            JTABLE_RULE_COL3_WIDTH);
        jTableRule.getSelectionModel().setSelectionMode(ListSelectionModel.
            SINGLE_SELECTION);
//        jTableRule.getModel().addTableModelListener(new TableModelListener() {
//
//            @Override
//            public void tableChanged(TableModelEvent e) {
//
//            }
//        });



        jTreeBuilding.getSelectionModel().setSelectionMode(TreeSelectionModel.
            SINGLE_TREE_SELECTION);
        jTreeBuilding.getSelectionModel().addTreeSelectionListener(new
            TreeSelectionListener() {

            @Override
            public void valueChanged(TreeSelectionEvent e) {

                ArrayList<ValueSource> sensorList = new ArrayList<ValueSource>();

                try {
                    if(jTreeBuilding.getLastSelectedPathComponent().getClass() ==
                        Room.class) {

                        jButtonAddToTree.setEnabled(false);
                        jButtonRemoveFromTree.setEnabled(true);
                        jButtonAddSensorToList.setEnabled(true);

                        for(int f = 0; f < ((BuildingTreeModel)jTreeBuilding.
                            getModel()).getBuilding().getFloorCount(); f++) {
                            for(int r = 0; r < ((BuildingTreeModel)jTreeBuilding.
                                getModel()).getBuilding().getFloor(f).getRoomCount
                                (); r++) {
                                if(((BuildingTreeModel)jTreeBuilding.getModel()).
                                    getBuilding().getFloor(f).getRoom(r) == e.
                                    getPath().getLastPathComponent()) {
                                    for(int s = 0; s < ((BuildingTreeModel)
                                        jTreeBuilding.getModel()).getBuilding().
                                        getFloor(f).getRoom(r).getSensorCount(); s
                                        ++) {
                                        sensorList.add(new ValueSource(((
                                            BuildingTreeModel)jTreeBuilding.
                                            getModel()).getBuilding(), f, r, s));
                                    }
                                }
                            }
                        }

                    } else if(jTreeBuilding.getLastSelectedPathComponent().getClass
                        () == Floor.class) {

                        jButtonAddToTree.setEnabled(true);
                        jButtonRemoveFromTree.setEnabled(true);
                        jButtonAddSensorToList.setEnabled(false);
```

```
                    for(int f = 0; f < ((BuildingTreeModel)jTreeBuilding.
                        getModel()).getBuilding().getFloorCount(); f++) {
                        if(((BuildingTreeModel)jTreeBuilding.getModel()).
                            getBuilding().getFloor(f) == e.getPath().
                            getLastPathComponent()) {
                            for(int r = 0; r < ((BuildingTreeModel)
                                jTreeBuilding.getModel()).getBuilding().
                                getFloor(f).getRoomCount(); r++) {
                                for(int s = 0; s < ((BuildingTreeModel)
                                    jTreeBuilding.getModel()).getBuilding().
                                    getFloor(f).getRoom(r).getSensorCount(); s
                                    ++)
                                    sensorList.add(new ValueSource(((
                                        BuildingTreeModel)jTreeBuilding.
                                        getModel()).getBuilding(), f, r, s));
                            }
                        }
                    }

                } else if(jTreeBuilding.getLastSelectedPathComponent().getClass
                    () == Building.class) {

                    jButtonAddToTree.setEnabled(true);
                    jButtonRemoveFromTree.setEnabled(false);
                    jButtonAddSensorToList.setEnabled(false);

                    for(int f = 0; f < ((BuildingTreeModel)jTreeBuilding.
                        getModel()).getBuilding().getFloorCount(); f++) {
                        for(int r = 0; r < ((BuildingTreeModel)jTreeBuilding.
                            getModel()).getBuilding().getFloor(f).getRoomCount
                            (); r++) {
                            for(int s = 0; s < ((BuildingTreeModel)
                                jTreeBuilding.getModel()).getBuilding().
                                getFloor(f).getRoom(r).getSensorCount(); s++)
                                sensorList.add(new ValueSource(((
                                    BuildingTreeModel)jTreeBuilding.getModel())
                                    .getBuilding(), f, r, s));
                        }
                    }

                }

                jTableSensor.setModel(new ValueSourceAbstractTableModel(
                    sensorList));
                jTableSensor.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
                jTableSensor.getColumnModel().getColumn(0).setPreferredWidth(
                    JTABLE_SENSOR_COL0_WIDTH);
                jTableSensor.getColumnModel().getColumn(1).setPreferredWidth(
                    JTABLE_SENSOR_COL1_WIDTH);
                jButtonRemoveSensorFromList.setEnabled(false);

            } catch(NullPointerException ex) {
                System.out.println(e.getSource() + "␣-->␣" + ex.toString());
            }
        }
    });


    jTableReaction.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
    jTableReaction.getColumnModel().getColumn(0).setPreferredWidth(
        JTABLE_REACTION_COL0_WIDTH);
    jTableReaction.getColumnModel().getColumn(1).setPreferredWidth(
        JTABLE_REACTION_COL1_WIDTH);
    jTableReaction.getColumnModel().getColumn(2).setPreferredWidth(
        JTABLE_REACTION_COL2_WIDTH);
    jTableReaction.getSelectionModel().setSelectionMode(ListSelectionModel.
        SINGLE_SELECTION);
```

```
        jTreeTupleCentre.getSelectionModel().setSelectionMode(TreeSelectionModel.
            SINGLE_TREE_SELECTION);
        jTreeTupleCentre.getSelectionModel().addTreeSelectionListener( new
            TreeSelectionListener() {

            @Override
            public void valueChanged(TreeSelectionEvent e) {

                ArrayList<Reaction> innerReactionList = new ArrayList<Reaction>();

                try {
                    if(jTreeTupleCentre.getLastSelectedPathComponent().getClass()
                        == TupleCentre.class) {

                        for(int t = 0; t < ((TupleCentreTreeModel)jTreeTupleCentre.
                            getModel()).getNode().getTupleCentreCount(); t++) {
                            if(((TupleCentreTreeModel)jTreeTupleCentre.getModel()).
                                getNode().getTupleCentre(t) == e.getPath().
                                getLastPathComponent())
                                innerReactionList.addAll(((TupleCentreTreeModel)
                                    jTreeTupleCentre.getModel()).getNode().
                                    getTupleCentre(t).getReactionList());
                        }

                        jButtonApplyToTupleCentre.setEnabled(true);
                        jButtonApplyToAllTupleCentre.setEnabled(false);

                    } else if(jTreeTupleCentre.getLastSelectedPathComponent().
                        getClass() == NodeEnvironment.class) {

                        for(int t = 0; t < ((TupleCentreTreeModel)jTreeTupleCentre.
                            getModel()).getNode().getTupleCentreCount(); t++) {
                            innerReactionList.addAll(((TupleCentreTreeModel)
                                jTreeTupleCentre.getModel()).getNode().
                                getTupleCentre(t).getReactionList());
                        }

                        jButtonApplyToTupleCentre.setEnabled(false);
                        jButtonApplyToAllTupleCentre.setEnabled(true);

                    }

                    jTableReaction.setModel(new ReactionAbstractTableModel(
                        innerReactionList));
                    jTableReaction.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
                    jTableReaction.getColumnModel().getColumn(0).setPreferredWidth(
                        JTABLE_REACTION_COL0_WIDTH);
                    jTableReaction.getColumnModel().getColumn(1).setPreferredWidth(
                        JTABLE_REACTION_COL1_WIDTH);
                    jTableReaction.getColumnModel().getColumn(2).setPreferredWidth(
                        JTABLE_REACTION_COL2_WIDTH);
                    jTableReaction.getSelectionModel().setSelectionMode(
                        ListSelectionModel.SINGLE_SELECTION);

                } catch(NullPointerException ex) {
                    System.out.println(e.getSource() + "␣-->␣" + ex.toString());
                }
            }
        });

    }

    private void initBuilding() {

        building = new Building("fundp-fi");

        building.addFloor(new Floor("floor0"));
        building.addFloor(new Floor("floor1"));

        building.getFloor(0).addRoom(new Room("rooma"));
```

```
        building.getFloor(0).addRoom(new Room("roomb"));

        building.getFloor(1).addRoom(new Room("roomc"));
        building.getFloor(1).addRoom(new Room("roomd"));
        building.getFloor(1).addRoom(new Room("roome"));

        building.getFloor(0).getRoom(0).addSensor(new Sensor("id0", SensorType.
            HUMIDITY));
        building.getFloor(0).getRoom(0).addSensor(new Sensor("id1", SensorType.
            TEMPERATURE));

        building.getFloor(0).getRoom(1).addSensor(new Sensor("id2", SensorType.
            TEMPERATURE));
        building.getFloor(0).getRoom(1).addSensor(new Sensor("id3", SensorType.
            PRESSURE));

        building.getFloor(1).getRoom(0).addSensor(new Sensor("id4", SensorType.
            LUMINOSITY));

        building.getFloor(1).getRoom(1).addSensor(new Sensor("id5", SensorType.
            LUMINOSITY));
        building.getFloor(1).getRoom(1).addSensor(new Sensor("id6", SensorType.
            TEMPERATURE));
        building.getFloor(1).getRoom(1).addSensor(new Sensor("id7", SensorType.
            TEMPERATURE));
        building.getFloor(1).getRoom(1).addSensor(new Sensor("id8", SensorType.
            HUMIDITY));

        building.getFloor(1).getRoom(2).addSensor(new Sensor("id9", SensorType.
            LUMINOSITY));
        building.getFloor(1).getRoom(2).addSensor(new Sensor("id10", SensorType.
            TEMPERATURE));

    }

    private void initRule() {

        defaultRuleList = new ArrayList<Rule>();

        // Compound
        defaultRuleList.add(new Rule());
        defaultRuleList.get(0).addCompoundRule(new CompoundRule());
        defaultRuleList.get(0).getCompoundRule(0).addSimpleRule(new SimpleRule());
        defaultRuleList.get(0).getCompoundRule(0).getSimpleRule(0).addRuleBody(new
            IntegerRuleBody(20000, RuleOperator.GREATER_THAN));
        defaultRuleList.get(0).getCompoundRule(0).getSimpleRule(0).setSource(new
            ValueSource(building, 1, 2, 0));
        defaultRuleList.get(0).getCompoundRule(0).addSimpleRule(new SimpleRule());
        defaultRuleList.get(0).getCompoundRule(0).getSimpleRule(1).addRuleBody(new
            IntegerRuleBody(55, RuleOperator.GREATER_THAN_OR_EQUAL_TO));
        defaultRuleList.get(0).getCompoundRule(0).getSimpleRule(1).setSource(new
            ValueSource(building, 1, 2, 1));
        defaultRuleList.get(0).setAction(new Action(ActionType.DIALOG_ALERT));



//        // Fixed <
//        defaultRuleList.add(new Rule());
//        defaultRuleList.get(0).addCompoundRule(new CompoundRule());
//        defaultRuleList.get(0).getCompoundRule(0).addSimpleRule(new SimpleRule())
    ;
//        defaultRuleList.get(0).getCompoundRule(0).getSimpleRule(0).addRuleBody(
    new IntegerRuleBody(50, RuleOperator.SMALLER_THAN));
//        defaultRuleList.get(0).getCompoundRule(0).getSimpleRule(0).setSource(new
    ValueSource(building, 0, 0, 0));
//        defaultRuleList.get(0).setAction(new Action(ActionType.DIALOG_ALERT));


//        // Fixed <=
//        defaultRuleList.add(new Rule());
```

```
//          defaultRuleList.get(1).addCompoundRule(new CompoundRule());
//          defaultRuleList.get(1).getCompoundRule(0).addSimpleRule(new SimpleRule())
    ;
//          defaultRuleList.get(1).getCompoundRule(0).getSimpleRule(0).addRuleBody(
    new IntegerRuleBody(58, RuleOperator.SMALLER_THAN_OR_EQUAL_TO));
//          defaultRuleList.get(1).getCompoundRule(0).getSimpleRule(0).setSource(new
    ValueSource(building, 0, 0, 1));
//          defaultRuleList.get(1).setAction(new Action(ActionType.DIALOG_ALERT));
//
//          // Fixed >
//          defaultRuleList.add(new Rule());
//          defaultRuleList.get(2).addCompoundRule(new CompoundRule());
//          defaultRuleList.get(2).getCompoundRule(0).addSimpleRule(new SimpleRule())
    ;
//          defaultRuleList.get(2).getCompoundRule(0).getSimpleRule(0).addRuleBody(
    new IntegerRuleBody(14, RuleOperator.GREATER_THAN));
//          defaultRuleList.get(2).getCompoundRule(0).getSimpleRule(0).setSource(new
    ValueSource(building, 0, 1, 0));
//          defaultRuleList.get(2).setAction(new Action(ActionType.PRINT_CONSOLE));
//
//          // Fixed >=
//          defaultRuleList.add(new Rule());
//          defaultRuleList.get(3).addCompoundRule(new CompoundRule());
//          defaultRuleList.get(3).getCompoundRule(0).addSimpleRule(new SimpleRule())
    ;
//          defaultRuleList.get(3).getCompoundRule(0).getSimpleRule(0).addRuleBody(
    new IntegerRuleBody(47, RuleOperator.GREATER_THAN_OR_EQUAL_TO));
//          defaultRuleList.get(3).getCompoundRule(0).getSimpleRule(0).setSource(new
    ValueSource(building, 0, 1, 1));
//          defaultRuleList.get(3).setAction(new Action(ActionType.DIALOG_ALERT));
//
//          // Fixed =
//          defaultRuleList.add(new Rule());
//          defaultRuleList.get(4).addCompoundRule(new CompoundRule());
//          defaultRuleList.get(4).getCompoundRule(0).addSimpleRule(new SimpleRule())
    ;
//          defaultRuleList.get(4).getCompoundRule(0).getSimpleRule(0).addRuleBody(
    new IntegerRuleBody(69, RuleOperator.EQUAL_TO));
//          defaultRuleList.get(4).getCompoundRule(0).getSimpleRule(0).setSource(new
    ValueSource(building, 0, 1, 0));
//          defaultRuleList.get(4).setAction(new Action(ActionType.DIALOG_ALERT));
//
//          // Fixed !=
//          defaultRuleList.add(new Rule());
//          defaultRuleList.get(5).addCompoundRule(new CompoundRule());
//          defaultRuleList.get(5).getCompoundRule(0).addSimpleRule(new SimpleRule())
    ;
//          defaultRuleList.get(5).getCompoundRule(0).getSimpleRule(0).addRuleBody(
    new IntegerRuleBody(36, RuleOperator.NOT_EQUAL_TO));
//          defaultRuleList.get(5).getCompoundRule(0).getSimpleRule(0).setSource(new
    ValueSource(building, 1, 1, 0));
//          defaultRuleList.get(5).setAction(new Action(ActionType.PRINT_CONSOLE));
//
//          // Range E
//          defaultRuleList.add(new Rule());
//          defaultRuleList.get(6).addCompoundRule(new CompoundRule());
//          defaultRuleList.get(6).getCompoundRule(0).addSimpleRule(new SimpleRule())
    ;
//          defaultRuleList.get(6).getCompoundRule(0).getSimpleRule(0).addRuleBody(
    new IntegerRuleBody(85, RuleOperator.GREATER_THAN));
//          defaultRuleList.get(6).getCompoundRule(0).getSimpleRule(0).addRuleBody(
    new IntegerRuleBody(125, RuleOperator.SMALLER_THAN_OR_EQUAL_TO));
//          defaultRuleList.get(6).getCompoundRule(0).getSimpleRule(0).
    setSpecialOperator(RuleOperator.INCLUDED_IN);
//          defaultRuleList.get(6).getCompoundRule(0).getSimpleRule(0).setSource(new
    ValueSource(building, 1, 1, 1));
//          defaultRuleList.get(6).setAction(new Action(ActionType.DIALOG_ALERT));
//
//          // Range !E
//          defaultRuleList.add(new Rule());
```

```
//          defaultRuleList.get(7).addCompoundRule(new CompoundRule());
//          defaultRuleList.get(7).getCompoundRule(0).addSimpleRule(new SimpleRule())
    ;
//          defaultRuleList.get(7).getCompoundRule(0).getSimpleRule(0).addRuleBody(
    new IntegerRuleBody(-52, RuleOperator.GREATER_THAN_OR_EQUAL_TO));
//          defaultRuleList.get(7).getCompoundRule(0).getSimpleRule(0).addRuleBody(
    new IntegerRuleBody(37, RuleOperator.SMALLER_THAN));
//          defaultRuleList.get(7).getCompoundRule(0).getSimpleRule(0).
    setSpecialOperator(RuleOperator.NOT_INCLUDED_IN);
//          defaultRuleList.get(7).getCompoundRule(0).getSimpleRule(0).setSource(new
    ValueSource(building, 0, 1, 1));
//          defaultRuleList.get(7).setAction(new Action(ActionType.PRINT_CONSOLE));
//
//          // Fixed =
//          defaultRuleList.add(new Rule());
//          defaultRuleList.get(8).addCompoundRule(new CompoundRule());
//          defaultRuleList.get(8).getCompoundRule(0).addSimpleRule(new SimpleRule())
    ;
//          defaultRuleList.get(8).getCompoundRule(0).getSimpleRule(0).addRuleBody(
    new StringRuleBody("hot", RuleOperator.EQUAL_TO));
//          defaultRuleList.get(8).getCompoundRule(0).getSimpleRule(0).setSource(new
    ValueSource(building, 0, 1, 0));
//          defaultRuleList.get(8).setAction(new Action(ActionType.DIALOG_ALERT));
//
//          // Fixed !=
//          defaultRuleList.add(new Rule());
//          defaultRuleList.get(9).addCompoundRule(new CompoundRule());
//          defaultRuleList.get(9).getCompoundRule(0).addSimpleRule(new SimpleRule())
    ;
//          defaultRuleList.get(9).getCompoundRule(0).getSimpleRule(0).addRuleBody(
    new StringRuleBody("cold", RuleOperator.NOT_EQUAL_TO));
//          defaultRuleList.get(9).getCompoundRule(0).getSimpleRule(0).setSource(new
    ValueSource(building, 1, 1, 0));
//          defaultRuleList.get(9).setAction(new Action(ActionType.PRINT_CONSOLE));

    }

    private String setReactionToTupleCentre(Boolean allTupleCentre) {

        String message = "";

        try {

            if(allTupleCentre) {

                for(int t = 0; t < ((TupleCentreTreeModel)jTreeTupleCentre.getModel
                    ()).getNode().getTupleCentreCount(); t++) {

                    int numberOfReactionSet = 0;

                    TupleCentre currentTupleCentre = (((TupleCentreTreeModel)
                        jTreeTupleCentre.getModel()).getNode().getTupleCentre(t));
                    TupleCentreId tcid = new TupleCentreId(currentTupleCentre.getId
                        ());

                    for(int i = 0; i < currentTupleCentre.getReactionCount(); i++)
                        {
                        if(!currentTupleCentre.getReaction(i).isSettled()) {
                            String previousSpec = ctx.getSpec(tcid);
                            ctx.setSpec(tcid, previousSpec + currentTupleCentre.
                                getReaction(i).toString());
                            (((TupleCentreTreeModel)jTreeTupleCentre.getModel()).
                                getNode().getTupleCentre(t)).getReaction(i).
                                setSettledStatus(true);
                            numberOfReactionSet++;
                        }
                    }

                    message += "Number of reactions set in the tuple centre '" +
                        currentTupleCentre.getId() + "' = " + numberOfReactionSet +
```

```
                        "\n";

                }

            } else {

                int numberOfReactionSet = 0;

                TupleCentre currentTupleCentre = (((TupleCentre)jTreeTupleCentre.
                    getSelectionPath().getLastPathComponent()));
                TupleCentreId tcid = new TupleCentreId(currentTupleCentre.getId());

                for(int i = 0; i < currentTupleCentre.getReactionCount(); i++) {
                    if(!currentTupleCentre.getReaction(i).isSettled()) {
                        String previousSpec = ctx.getSpec(tcid);
                        System.out.println(currentTupleCentre.getReaction(i).
                            toString());
                        ctx.setSpec(tcid, previousSpec + currentTupleCentre.
                            getReaction(i).toString());
                        (((TupleCentre)jTreeTupleCentre.getSelectionPath().
                            getLastPathComponent())).getReaction(i).
                            setSettledStatus(true);
                        numberOfReactionSet++;
                    }
                }

                message = "Number of reactions set in the tuple centre '" + ((
                    TupleCentre)jTreeTupleCentre.getSelectionPath().
                    getLastPathComponent()).getId() + "' = " + numberOfReactionSet;

            }

            //ctx.exit();

        } catch (InvalidTupleCentreIdException ex) {
            System.err.println(ex);
        } catch (OperationNotAllowedException ex) {
            System.err.println(ex);
        } catch (UnreachableNodeException ex) {
            System.err.println(ex);
        } catch (InvalidSpecificationException ex) {
            System.err.println(ex);
        }

        return message;

    }

    private void resetAllTupleCentres() {
        try {

            for(int t = 0; t < ((TupleCentreTreeModel)jTreeTupleCentre.getModel()).
                getNode().getTupleCentreCount(); t++) {

                TupleCentre currentTupleCentre = (((TupleCentreTreeModel)
                    jTreeTupleCentre.getModel()).getNode().getTupleCentre(t));
                TupleCentreId tcid = new TupleCentreId(currentTupleCentre.getId());

                // Remove all specification
                for(int i = 0; i < currentTupleCentre.getReactionCount(); i++) {
                    ctx.setSpec(tcid, "");
                    (((TupleCentreTreeModel)jTreeTupleCentre.getModel()).getNode().
                        getTupleCentre(t)).getReaction(i).setSettledStatus(false);
                }

                // Remove all tuples
                LogicTuple req = null;
                do {
                    req = ctx.inp(tcid, new LogicTuple(new Var()));
                } while(req != null);
```

```
            }

        } catch (InvalidTupleCentreIdException ex) {
            System.err.println(ex);
        } catch (OperationNotAllowedException ex) {
            System.err.println(ex);
        } catch (UnreachableNodeException ex) {
            System.err.println(ex);
        } catch (InvalidSpecificationException ex) {
            System.err.println(ex);
        }
    }

    private ArrayList<String> getTupleCentreListFromBuilding(Building building) {

        ArrayList<String> tcList = new ArrayList<String>();

        for(int f = 0; f < building.getFloorCount(); f++) {
            for(int r = 0; r < building.getFloor(f).getRoomCount(); r++) {
                tcList.add((building.getFloor(f) + "_" + building.getFloor(f).
                    getRoom(r)));
            }
        }

        return tcList;
    }


    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:
        initComponents
    private void initComponents() {

        jTabbedPane = new javax.swing.JTabbedPane();
        jPanelTabUser = new javax.swing.JPanel();
        jScrollPaneTree = new javax.swing.JScrollPane();
        jTreeBuilding = new javax.swing.JTree();
        jLabelBuildingOfInterest = new javax.swing.JLabel();
        jButtonAddToTree = new javax.swing.JButton();
        jButtonModifyInTree = new javax.swing.JButton();
        jButtonRemoveFromTree = new javax.swing.JButton();
        jScrollPaneSensorTable = new javax.swing.JScrollPane();
        jTableSensor = new javax.swing.JTable();
        jLabelSensorList = new javax.swing.JLabel();
        jScrollPaneRuleTable = new javax.swing.JScrollPane();
        jTableRule = new javax.swing.JTable();
        jLabelRuleList = new javax.swing.JLabel();
        jButtonAddSensorToList = new javax.swing.JButton();
        jButtonEditSensorInList = new javax.swing.JButton();
        jButtonRemoveSensorFromList = new javax.swing.JButton();
        jSpinnerSamplingPeriod = new javax.swing.JSpinner();
        jLabelSamplingPeriod = new javax.swing.JLabel();
        jLabelUnit = new javax.swing.JLabel();
        jButtonAddRuleToList = new javax.swing.JButton();
        jButtonEditRuleInList = new javax.swing.JButton();
        jButtonRemoveRuleFromList = new javax.swing.JButton();
        jPanelTabDeveloper = new javax.swing.JPanel();
        jLabelTupleCentres = new javax.swing.JLabel();
        jScrollPaneTupleCentre = new javax.swing.JScrollPane();
        jTreeTupleCentre = new javax.swing.JTree();
        jButtonResetTupleCentre = new javax.swing.JButton();
        jButtonApplyToAllTupleCentre = new javax.swing.JButton();
        jButtonApplyToTupleCentre = new javax.swing.JButton();
        jLabelReactions = new javax.swing.JLabel();
```

```
        jScrollPaneReactions = new javax.swing.JScrollPane();
        jTableReaction = new javax.swing.JTable();
        jLabelMonitor = new javax.swing.JLabel();
        jScrollPaneMonitoring = new javax.swing.JScrollPane();
        jTextAreaMonitoring = new javax.swing.JTextArea();
        jButtonPlayPause = new javax.swing.JButton();
        jPanelTucsonNode = new javax.swing.JPanel();
        jButtonLaunchTucsonNode = new javax.swing.JButton();
        jTextFieldTucsonNodeStatus = new javax.swing.JTextField();
        jLabelTucsonNodeStatus = new javax.swing.JLabel();
        jButtonUpdateRuleList = new javax.swing.JButton();
        jMenuBar = new javax.swing.JMenuBar();
        jMenuFile = new javax.swing.JMenu();
        jMenuEdit = new javax.swing.JMenu();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Environment Manager");
        setLocationByPlatform(true);
        setResizable(false);

        jTabbedPane.addChangeListener(new javax.swing.event.ChangeListener() {
            public void stateChanged(javax.swing.event.ChangeEvent evt) {
                jTabbedPaneStateChanged(evt);
            }
        });

        jTreeBuilding.setModel(new BuildingTreeModel(building));
        jScrollPaneTree.setViewportView(jTreeBuilding);

        jLabelBuildingOfInterest.setFont(new java.awt.Font("Tahoma", 1, 11));
        jLabelBuildingOfInterest.setText("Building of Interest :");

        jButtonAddToTree.setText("Add Element");
        jButtonAddToTree.setEnabled(false);
        jButtonAddToTree.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonAddToTreeActionPerformed(evt);
            }
        });

        jButtonModifyInTree.setText("Edit Element");
        jButtonModifyInTree.setEnabled(false);

        jButtonRemoveFromTree.setText("Remove Element");
        jButtonRemoveFromTree.setEnabled(false);
        jButtonRemoveFromTree.addActionListener(new java.awt.event.ActionListener()
             {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonRemoveFromTreeActionPerformed(evt);
            }
        });

        jTableSensor.setModel(new ValueSourceAbstractTableModel());
        jTableSensor.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jTableSensorMouseClicked(evt);
            }
        });
        jScrollPaneSensorTable.setViewportView(jTableSensor);

        jLabelSensorList.setFont(new java.awt.Font("Tahoma", 1, 11));
        jLabelSensorList.setText("Sensors' List :");

        jTableRule.setModel(new RuleAbstractTableModel(defaultRuleList));
        jTableRule.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        jTableRule.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jTableRuleMouseClicked(evt);
            }
        });
```

```
        jScrollPaneRuleTable.setViewportView(jTableRule);

        jLabelRuleList.setFont(new java.awt.Font("Tahoma", 1, 11));
        jLabelRuleList.setText("Rules' List :");

        jButtonAddSensorToList.setText("Add Sensor");
        jButtonAddSensorToList.setEnabled(false);
        jButtonAddSensorToList.addActionListener(new java.awt.event.ActionListener
            () {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonAddSensorToListActionPerformed(evt);
            }
        });

        jButtonEditSensorInList.setText("Edit Sensor");
        jButtonEditSensorInList.setEnabled(false);

        jButtonRemoveSensorFromList.setText("Remove Sensor");
        jButtonRemoveSensorFromList.setEnabled(false);
        jButtonRemoveSensorFromList.addActionListener(new java.awt.event.
            ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonRemoveSensorFromListActionPerformed(evt);
            }
        });

        jSpinnerSamplingPeriod.setModel(new javax.swing.SpinnerNumberModel(Integer.
            valueOf(5), Integer.valueOf(1), null, Integer.valueOf(1)));

        jLabelSamplingPeriod.setText("Sampling Period :");

        jLabelUnit.setText("[s]");

        jButtonAddRuleToList.setText("Add Rule");
        jButtonAddRuleToList.addActionListener(new java.awt.event.ActionListener()
            {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonAddRuleToListActionPerformed(evt);
            }
        });

        jButtonEditRuleInList.setText("Edit Rule");
        jButtonEditRuleInList.setEnabled(false);

        jButtonRemoveRuleFromList.setText("Remove Rule");
        jButtonRemoveRuleFromList.setEnabled(false);
        jButtonRemoveRuleFromList.addActionListener(new java.awt.event.
            ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonRemoveRuleFromListActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanelTabUserLayout = new javax.swing.GroupLayout(
            jPanelTabUser);
        jPanelTabUser.setLayout(jPanelTabUserLayout);
        jPanelTabUserLayout.setHorizontalGroup(
            jPanelTabUserLayout.createParallelGroup(javax.swing.GroupLayout.
                Alignment.LEADING)
            .addGroup(jPanelTabUserLayout.createSequentialGroup()
                .addGroup(jPanelTabUserLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.LEADING)
                    .addGroup(jPanelTabUserLayout.createSequentialGroup()
                        .addContainerGap()
                        .addGroup(jPanelTabUserLayout.createParallelGroup(javax.
                            swing.GroupLayout.Alignment.CENTER)
                            .addComponent(jScrollPaneTree, javax.swing.GroupLayout.
                                PREFERRED_SIZE, 158, javax.swing.GroupLayout.
                                PREFERRED_SIZE)
```

```
                        .addComponent(jButtonAddToTree, javax.swing.GroupLayout
                            .PREFERRED_SIZE, 158, javax.swing.GroupLayout.
                            PREFERRED_SIZE)
                        .addComponent(jButtonModifyInTree, javax.swing.
                            GroupLayout.PREFERRED_SIZE, 158, javax.swing.
                            GroupLayout.PREFERRED_SIZE)
                        .addComponent(jButtonRemoveFromTree, javax.swing.
                            GroupLayout.PREFERRED_SIZE, 158, javax.swing.
                            GroupLayout.PREFERRED_SIZE)))
                .addGroup(jPanelTabUserLayout.createSequentialGroup()
                    .addGap(10, 10, 10)
                    .addComponent(jLabelBuildingOfInterest)))
            .addGap(18, 18, 18)
            .addGroup(jPanelTabUserLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.LEADING, false)
                .addComponent(jScrollPaneSensorTable, 0, 0, Short.MAX_VALUE)
                .addComponent(jLabelSensorList)
                .addComponent(jButtonRemoveSensorFromList, javax.swing.
                    GroupLayout.PREFERRED_SIZE, 158, javax.swing.GroupLayout.
                    PREFERRED_SIZE)
                .addComponent(jButtonEditSensorInList, javax.swing.GroupLayout.
                    PREFERRED_SIZE, 158, javax.swing.GroupLayout.PREFERRED_SIZE
                    )
                .addComponent(jButtonAddSensorToList, javax.swing.GroupLayout.
                    PREFERRED_SIZE, 158, javax.swing.GroupLayout.PREFERRED_SIZE
                    ))
            .addGroup(jPanelTabUserLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.TRAILING)
                .addGroup(jPanelTabUserLayout.createSequentialGroup()
                    .addGap(18, 18, 18)
                    .addGroup(jPanelTabUserLayout.createParallelGroup(javax.
                        swing.GroupLayout.Alignment.LEADING)
                        .addGroup(jPanelTabUserLayout.createSequentialGroup()
                            .addComponent(jLabelRuleList)
                            .addPreferredGap(javax.swing.LayoutStyle.
                                ComponentPlacement.RELATED, javax.swing.
                                GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(jLabelSamplingPeriod)
                            .addPreferredGap(javax.swing.LayoutStyle.
                                ComponentPlacement.RELATED)
                            .addComponent(jSpinnerSamplingPeriod, javax.swing.
                                GroupLayout.PREFERRED_SIZE, 83, javax.swing.
                                GroupLayout.PREFERRED_SIZE)
                            .addPreferredGap(javax.swing.LayoutStyle.
                                ComponentPlacement.RELATED)
                            .addComponent(jLabelUnit))
                        .addComponent(jScrollPaneRuleTable, javax.swing.
                            GroupLayout.DEFAULT_SIZE, 503, Short.MAX_VALUE)))
                .addGroup(jPanelTabUserLayout.createSequentialGroup()
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                        .RELATED)
                    .addGroup(jPanelTabUserLayout.createParallelGroup(javax.
                        swing.GroupLayout.Alignment.LEADING, false)
                        .addComponent(jButtonRemoveRuleFromList, javax.swing.
                            GroupLayout.PREFERRED_SIZE, 158, javax.swing.
                            GroupLayout.PREFERRED_SIZE)
                        .addComponent(jButtonEditRuleInList, javax.swing.
                            GroupLayout.PREFERRED_SIZE, 158, javax.swing.
                            GroupLayout.PREFERRED_SIZE)
                        .addComponent(jButtonAddRuleToList, javax.swing.
                            GroupLayout.PREFERRED_SIZE, 158, javax.swing.
                            GroupLayout.PREFERRED_SIZE))))
            .addContainerGap())
    );
    jPanelTabUserLayout.setVerticalGroup(
        jPanelTabUserLayout.createParallelGroup(javax.swing.GroupLayout.
            Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
            jPanelTabUserLayout.createSequentialGroup()
            .addContainerGap()
```

```
                .addGroup(jPanelTabUserLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabelBuildingOfInterest)
                    .addComponent(jLabelRuleList)
                    .addComponent(jLabelUnit)
                    .addComponent(jSpinnerSamplingPeriod, javax.swing.GroupLayout.
                        PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax
                        .swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabelSamplingPeriod)
                    .addComponent(jLabelSensorList))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
                    )
                .addGroup(jPanelTabUserLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.LEADING)
                    .addComponent(jScrollPaneRuleTable, javax.swing.GroupLayout.
                        DEFAULT_SIZE, 394, Short.MAX_VALUE)
                    .addComponent(jScrollPaneSensorTable, javax.swing.GroupLayout.
                        Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
                        394, Short.MAX_VALUE)
                    .addComponent(jScrollPaneTree, javax.swing.GroupLayout.
                        Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
                        394, Short.MAX_VALUE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
                    )
                .addGroup(jPanelTabUserLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.LEADING)
                    .addGroup(jPanelTabUserLayout.createParallelGroup(javax.swing.
                        GroupLayout.Alignment.LEADING)
                        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                            jPanelTabUserLayout.createSequentialGroup()
                            .addComponent(jButtonAddToTree)
                            .addPreferredGap(javax.swing.LayoutStyle.
                                ComponentPlacement.RELATED)
                            .addComponent(jButtonModifyInTree)
                            .addPreferredGap(javax.swing.LayoutStyle.
                                ComponentPlacement.RELATED)
                            .addComponent(jButtonRemoveFromTree))
                        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                            jPanelTabUserLayout.createSequentialGroup()
                            .addComponent(jButtonAddSensorToList)
                            .addPreferredGap(javax.swing.LayoutStyle.
                                ComponentPlacement.RELATED)
                            .addComponent(jButtonEditSensorInList)
                            .addPreferredGap(javax.swing.LayoutStyle.
                                ComponentPlacement.RELATED)
                            .addComponent(jButtonRemoveSensorFromList)))
                    .addGroup(jPanelTabUserLayout.createSequentialGroup()
                        .addComponent(jButtonAddRuleToList)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                            .RELATED)
                        .addComponent(jButtonEditRuleInList)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                            .RELATED)
                        .addComponent(jButtonRemoveRuleFromList)))
                .addContainerGap())
        );

        jTabbedPane.addTab("User", jPanelTabUser);

        jLabelTupleCentres.setFont(new java.awt.Font("Tahoma", 1, 11));
        jLabelTupleCentres.setText("Tuple Centres :");

        jTreeTupleCentre.setModel(new TupleCentreTreeModel());
        jScrollPaneTupleCentre.setViewportView(jTreeTupleCentre);

        jButtonResetTupleCentre.setText("Reset Tuple Centre");
        jButtonResetTupleCentre.addActionListener(new java.awt.event.ActionListener
            () {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonResetTupleCentreActionPerformed(evt);
```

```
        }
    });

    jButtonApplyToAllTupleCentre.setText("Apply Reactions to All");
    jButtonApplyToAllTupleCentre.setEnabled(false);
    jButtonApplyToAllTupleCentre.addActionListener(new java.awt.event.
        ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonApplyToAllTupleCentreActionPerformed(evt);
        }
    });

    jButtonApplyToTupleCentre.setText("Apply Reactions to TC");
    jButtonApplyToTupleCentre.setEnabled(false);
    jButtonApplyToTupleCentre.addActionListener(new java.awt.event.
        ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonApplyToTupleCentreActionPerformed(evt);
        }
    });

    jLabelReactions.setFont(new java.awt.Font("Tahoma", 1, 11));
    jLabelReactions.setText("Reactions :");

    jTableReaction.setModel(new ReactionAbstractTableModel());
    jScrollPaneReactions.setViewportView(jTableReaction);

    jLabelMonitor.setFont(new java.awt.Font("Tahoma", 1, 11));
    jLabelMonitor.setText("Monitor :");

    jTextAreaMonitoring.setColumns(20);
    jTextAreaMonitoring.setEditable(false);
    jTextAreaMonitoring.setFont(new java.awt.Font("Consolas", 0, 10));
    jTextAreaMonitoring.setRows(5);
    jScrollPaneMonitoring.setViewportView(jTextAreaMonitoring);

    jButtonPlayPause.setIcon(new javax.swing.ImageIcon(getClass().getResource("
        /be/woine/thesis/utilz/Play-Normal-icon.png"))); // NOI18N
    jButtonPlayPause.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonPlayPauseActionPerformed(evt);
        }
    });

    jPanelTucsonNode.setBorder(javax.swing.BorderFactory.createTitledBorder(
        null, "TuCSoN Node", javax.swing.border.TitledBorder.
        DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.DEFAULT_POSITION
        , new java.awt.Font("Tahoma", 1, 11), new java.awt.Color(0, 0, 255)));
        // NOI18N

    jButtonLaunchTucsonNode.setText("Launch Node");
    jButtonLaunchTucsonNode.addActionListener(new java.awt.event.ActionListener
        () {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonLaunchTucsonNodeActionPerformed(evt);
        }
    });

    jTextFieldTucsonNodeStatus.setEditable(false);
    jTextFieldTucsonNodeStatus.setFont(new java.awt.Font("Tahoma", 2, 11));

    jLabelTucsonNodeStatus.setText("Status :");

    jButtonUpdateRuleList.setText("Update List of Rules");
    jButtonUpdateRuleList.setEnabled(false);
    jButtonUpdateRuleList.addActionListener(new java.awt.event.ActionListener()
         {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonUpdateRuleListActionPerformed(evt);
        }
```

```
        });

        javax.swing.GroupLayout jPanelTucsonNodeLayout = new javax.swing.
            GroupLayout(jPanelTucsonNode);
        jPanelTucsonNode.setLayout(jPanelTucsonNodeLayout);
        jPanelTucsonNodeLayout.setHorizontalGroup(
            jPanelTucsonNodeLayout.createParallelGroup(javax.swing.GroupLayout.
                Alignment.LEADING)
            .addGroup(jPanelTucsonNodeLayout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanelTucsonNodeLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.LEADING)
                    .addGroup(jPanelTucsonNodeLayout.createSequentialGroup()
                        .addComponent(jLabelTucsonNodeStatus)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                            .RELATED)
                        .addComponent(jTextFieldTucsonNodeStatus, javax.swing.
                            GroupLayout.DEFAULT_SIZE, 214, Short.MAX_VALUE))
                    .addGroup(jPanelTucsonNodeLayout.createSequentialGroup()
                        .addComponent(jButtonLaunchTucsonNode)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                            .RELATED, 32, Short.MAX_VALUE)
                        .addComponent(jButtonUpdateRuleList)))
                .addContainerGap())
        );
        jPanelTucsonNodeLayout.setVerticalGroup(
            jPanelTucsonNodeLayout.createParallelGroup(javax.swing.GroupLayout.
                Alignment.LEADING)
            .addGroup(jPanelTucsonNodeLayout.createSequentialGroup()
                .addGap(31, 31, 31)
                .addGroup(jPanelTucsonNodeLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.BASELINE)
                    .addComponent(jButtonLaunchTucsonNode)
                    .addComponent(jButtonUpdateRuleList))
                .addGap(38, 38, 38)
                .addGroup(jPanelTucsonNodeLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.BASELINE)
                    .addComponent(jTextFieldTucsonNodeStatus, javax.swing.
                        GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.
                        DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabelTucsonNodeStatus))
                .addContainerGap(61, Short.MAX_VALUE))
        );

        javax.swing.GroupLayout jPanelTabDeveloperLayout = new javax.swing.
            GroupLayout(jPanelTabDeveloper);
        jPanelTabDeveloper.setLayout(jPanelTabDeveloperLayout);
        jPanelTabDeveloperLayout.setHorizontalGroup(
            jPanelTabDeveloperLayout.createParallelGroup(javax.swing.GroupLayout.
                Alignment.LEADING)
            .addGroup(jPanelTabDeveloperLayout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanelTabDeveloperLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.LEADING, false)
                    .addComponent(jScrollPaneTupleCentre, javax.swing.GroupLayout.
                        DEFAULT_SIZE, 134, Short.MAX_VALUE)
                    .addComponent(jLabelTupleCentres)
                    .addComponent(jButtonResetTupleCentre, javax.swing.GroupLayout.
                        DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                        MAX_VALUE)
                    .addComponent(jButtonApplyToAllTupleCentre, javax.swing.
                        GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.
                        DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(jButtonApplyToTupleCentre, javax.swing.
                        GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.
                        DEFAULT_SIZE, Short.MAX_VALUE))
                .addGap(18, 18, 18)
                .addGroup(jPanelTabDeveloperLayout.createParallelGroup(javax.swing.
                    GroupLayout.Alignment.LEADING)
```

```
                .addComponent(jScrollPaneReactions, javax.swing.GroupLayout.
                    DEFAULT_SIZE, 698, Short.MAX_VALUE)
                .addComponent(jLabelReactions)
                .addGroup(jPanelTabDeveloperLayout.createSequentialGroup()
                    .addGroup(jPanelTabDeveloperLayout.createParallelGroup(
                        javax.swing.GroupLayout.Alignment.LEADING)
                        .addGroup(jPanelTabDeveloperLayout.
                            createSequentialGroup()
                            .addComponent(jScrollPaneMonitoring, javax.swing.
                                GroupLayout.PREFERRED_SIZE, 362, javax.swing.
                                GroupLayout.PREFERRED_SIZE)
                            .addPreferredGap(javax.swing.LayoutStyle.
                                ComponentPlacement.RELATED)
                            .addComponent(jButtonPlayPause, javax.swing.
                                GroupLayout.PREFERRED_SIZE, 24, javax.swing.
                                GroupLayout.PREFERRED_SIZE))
                        .addComponent(jLabelMonitor))
                    .addGap(18, 18, 18)
                    .addComponent(jPanelTucsonNode, javax.swing.GroupLayout.
                        DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                        Short.MAX_VALUE)))
            .addContainerGap())
    );
    jPanelTabDeveloperLayout.setVerticalGroup(
        jPanelTabDeveloperLayout.createParallelGroup(javax.swing.GroupLayout.
            Alignment.LEADING)
        .addGroup(jPanelTabDeveloperLayout.createSequentialGroup()
            .addContainerGap()
            .addGroup(jPanelTabDeveloperLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.BASELINE)
                .addComponent(jLabelTupleCentres)
                .addComponent(jLabelReactions))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
                )
            .addGroup(jPanelTabDeveloperLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.LEADING)
                .addGroup(jPanelTabDeveloperLayout.createSequentialGroup()
                    .addComponent(jScrollPaneTupleCentre, javax.swing.
                        GroupLayout.DEFAULT_SIZE, 400, Short.MAX_VALUE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                        .RELATED)
                    .addComponent(jButtonApplyToTupleCentre)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                        .RELATED)
                    .addComponent(jButtonApplyToAllTupleCentre)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                        .RELATED)
                    .addComponent(jButtonResetTupleCentre))
                .addGroup(jPanelTabDeveloperLayout.createSequentialGroup()
                    .addComponent(jScrollPaneReactions, javax.swing.GroupLayout
                        .PREFERRED_SIZE, 269, javax.swing.GroupLayout.
                        PREFERRED_SIZE)
                    .addGap(18, 18, 18)
                    .addGroup(jPanelTabDeveloperLayout.createParallelGroup(
                        javax.swing.GroupLayout.Alignment.TRAILING)
                        .addGroup(jPanelTabDeveloperLayout.
                            createSequentialGroup()
                            .addComponent(jLabelMonitor)
                            .addPreferredGap(javax.swing.LayoutStyle.
                                ComponentPlacement.RELATED)
                            .addGroup(jPanelTabDeveloperLayout.
                                createParallelGroup(javax.swing.GroupLayout.
                                Alignment.LEADING)
                                .addComponent(jButtonPlayPause, javax.swing.
                                    GroupLayout.PREFERRED_SIZE, 24, javax.swing
                                    .GroupLayout.PREFERRED_SIZE)
                                .addComponent(jScrollPaneMonitoring, javax.
                                    swing.GroupLayout.DEFAULT_SIZE, 180, Short.
                                    MAX_VALUE)))))
```

```
                                .addComponent(jPanelTucsonNode, javax.swing.GroupLayout
                                    .DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE
                                    , Short.MAX_VALUE))))
                .addContainerGap())
        );

        jTabbedPane.addTab("Developer", jPanelTabDeveloper);

        jMenuFile.setText("File");
        jMenuBar.add(jMenuFile);

        jMenuEdit.setText("Edit");
        jMenuBar.add(jMenuEdit);

        setJMenuBar(jMenuBar);

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane
            ());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jTabbedPane, javax.swing.GroupLayout.DEFAULT_SIZE,
                    880, Short.MAX_VALUE)
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jTabbedPane, javax.swing.GroupLayout.DEFAULT_SIZE,
                    557, Short.MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jButtonAddRuleToListActionPerformed(java.awt.event.ActionEvent evt
        ) {//GEN-FIRST:event_jButtonAddRuleToListActionPerformed

        JDialogRuleChooser dialogRule = new JDialogRuleChooser(this, true);
        dialogRule.setLocationByPlatform(true);
        dialogRule.setVisible(true);

        if(dialogRule.getReturnValue() == JOptionPane.OK_OPTION) {

            /* Fixed Rule */
            if(dialogRule.getRuleType() == RuleType.FIXED) {

                JDialogSimpleRuleFixed dialogFixed = new JDialogSimpleRuleFixed(((
                    BuildingTreeModel)jTreeBuilding.getModel()).getBuilding(), this
                    , true);
                dialogFixed.setLocationByPlatform(true);
                dialogFixed.setVisible(true);

                if(dialogFixed.getReturnValue() == JOptionPane.OK_OPTION) {

                    Rule newRule = new Rule();
                    newRule.addCompoundRule(new CompoundRule());
                    newRule.getCompoundRule(0).addSimpleRule(dialogFixed.
                        getSimpleRule());
                    newRule.setAction(new Action(dialogRule.getActionType()));

                    ((RuleAbstractTableModel)jTableRule.getModel()).addRule(newRule
                        );
                }

            /* Range Rule */
```

```
        } else if(dialogRule.getRuleType() == RuleType.RANGE) {

            JDialogSimpleRuleRange dialogRange = new JDialogSimpleRuleRange(((
                BuildingTreeModel)jTreeBuilding.getModel()).getBuilding(), this
                , true);
            dialogRange.setLocationByPlatform(true);
            dialogRange.setVisible(true);

            if(dialogRange.getReturnValue() == JOptionPane.OK_OPTION) {

                Rule newRule = new Rule();
                newRule.addCompoundRule(new CompoundRule());
                newRule.getCompoundRule(0).addSimpleRule(dialogRange.
                    getSimpleRule());
                newRule.setAction(new Action(dialogRule.getActionType()));

                ((RuleAbstractTableModel)jTableRule.getModel()).addRule(newRule
                    );
            }

        /* Compound Rule */
        } else if(dialogRule.getRuleType() == RuleType.COMPOUND) {

            JDialogCompoundRule dialogCompound = new JDialogCompoundRule(((
                BuildingTreeModel)jTreeBuilding.getModel()).getBuilding(), this
                , true);
            dialogCompound.setLocationByPlatform(true);
            dialogCompound.setVisible(true);

            if(dialogCompound.getReturnValue() == JOptionPane.OK_OPTION) {

                Rule newRule = new Rule();
                newRule.addCompoundRule(dialogCompound.getCompoundRule());
                newRule.setAction(new Action(dialogRule.getActionType()));

                ((RuleAbstractTableModel)jTableRule.getModel()).addRule(newRule
                    );
            }

        /* Sequence Rule */
        } else if(dialogRule.getRuleType() == RuleType.SEQUENCE) {

            JDialogSequenceRule dialogSequence = new JDialogSequenceRule(((
                BuildingTreeModel)jTreeBuilding.getModel()).getBuilding(), this
                , true);
            dialogSequence.setLocationByPlatform(true);
            dialogSequence.setVisible(true);

            if(dialogSequence.getReturnValue() == JOptionPane.OK_OPTION) {

                Rule newRule = dialogSequence.getSequenceRule();
                newRule.setAction(new Action(dialogRule.getActionType()));

                ((RuleAbstractTableModel)jTableRule.getModel()).addRule(newRule
                    );
            }

        }

        jTableRule.getColumnModel().getColumn(0).setPreferredWidth(
            JTABLE_RULE_COL0_WIDTH);
        jTableRule.getColumnModel().getColumn(1).setPreferredWidth(
            JTABLE_RULE_COL1_WIDTH);
        jTableRule.getColumnModel().getColumn(2).setPreferredWidth(
            JTABLE_RULE_COL2_WIDTH);
        jTableRule.getColumnModel().getColumn(3).setPreferredWidth(
            JTABLE_RULE_COL3_WIDTH);

    }
```

```
}//GEN-LAST:event_jButtonAddRuleToListActionPerformed

private void jButtonRemoveRuleFromListActionPerformed(java.awt.event.
    ActionEvent evt) {//GEN-FIRST:
    event_jButtonRemoveRuleFromListActionPerformed
    if(jTableRule.getSelectedRow() >= 0) {
        ((RuleAbstractTableModel)jTableRule.getModel()).removeRule(jTableRule.
            getSelectedRow());

        jTableRule.getColumnModel().getColumn(0).setPreferredWidth(
            JTABLE_RULE_COL0_WIDTH);
        jTableRule.getColumnModel().getColumn(1).setPreferredWidth(
            JTABLE_RULE_COL1_WIDTH);
        jTableRule.getColumnModel().getColumn(2).setPreferredWidth(
            JTABLE_RULE_COL2_WIDTH);
        jTableRule.getColumnModel().getColumn(3).setPreferredWidth(
            JTABLE_RULE_COL3_WIDTH);
    }
    jButtonRemoveRuleFromList.setEnabled(false);
}//GEN-LAST:event_jButtonRemoveRuleFromListActionPerformed

private void jButtonAddToTreeActionPerformed(java.awt.event.ActionEvent evt) {
    //GEN-FIRST:event_jButtonAddToTreeActionPerformed

    try {
        if(jTreeBuilding.getLastSelectedPathComponent().getClass() == Building.
            class) {
            /* Add Floor */
            JDialogTreeElement dialogElement = new JDialogTreeElement("Floor",
                this, true);
            dialogElement.setLocationByPlatform(true);
            dialogElement.setVisible(true);

            if(dialogElement.getReturnValue() == JOptionPane.OK_OPTION) {
                ((BuildingTreeModel)jTreeBuilding.getModel()).addFloor(new
                    Floor(dialogElement.getElementName()));

            }

        } else if(jTreeBuilding.getLastSelectedPathComponent().getClass() ==
            Floor.class) {
            /* Add Room */
            JDialogTreeElement dialogElement = new JDialogTreeElement("Room",
                this, true);
            dialogElement.setLocationByPlatform(true);
            dialogElement.setVisible(true);

            if(dialogElement.getReturnValue() == JOptionPane.OK_OPTION) {
                ((BuildingTreeModel)jTreeBuilding.getModel()).addRoom(((
                    BuildingTreeModel)jTreeBuilding.getModel()).getIndexOfChild
                    (((BuildingTreeModel)jTreeBuilding.getModel()).getBuilding
                    (), jTreeBuilding.getLastSelectedPathComponent()), new Room
                    (dialogElement.getElementName()));
            }
        }
    } catch(NullPointerException ex) {
        System.out.println("-->" + ex.toString());
    }



}//GEN-LAST:event_jButtonAddToTreeActionPerformed

private void jButtonAddSensorToListActionPerformed(java.awt.event.ActionEvent
    evt) {//GEN-FIRST:event_jButtonAddSensorToListActionPerformed

    try {
        if(jTreeBuilding.getLastSelectedPathComponent().getClass() == Room.
            class) {
```

```
                JDialogSensor dialog = new JDialogSensor(this, true);
                dialog.setLocationByPlatform(true);
                dialog.setVisible(true);

                if(dialog.getReturnValue() == JOptionPane.OK_OPTION) {

                    int floorIndex = (((BuildingTreeModel)jTreeBuilding.getModel())
                        .getIndexOfChild(((BuildingTreeModel)jTreeBuilding.getModel
                        ()).getBuilding(), jTreeBuilding.getSelectionPath().
                        getParentPath().getLastPathComponent()));
                    int roomIndex = (((BuildingTreeModel)jTreeBuilding.getModel()).
                        getIndexOfChild(jTreeBuilding.getSelectionPath().
                        getParentPath().getLastPathComponent(), jTreeBuilding.
                        getSelectionPath().getLastPathComponent()));

                    ((BuildingTreeModel)jTreeBuilding.getModel()).addSensor(
                        floorIndex, roomIndex, new Sensor(dialog.getSensorId(),
                        dialog.getSensorType()));

                    ArrayList<Sensor> sensorList = ((BuildingTreeModel)
                        jTreeBuilding.getModel()).getSensorList(floorIndex,
                        roomIndex);
                    ArrayList<ValueSource> valueSourceList = new ArrayList<
                        ValueSource>();

                    for(int s = 0; s < sensorList.size(); s++)
                        valueSourceList.add(new ValueSource(((BuildingTreeModel)
                            jTreeBuilding.getModel()).getBuilding(), floorIndex,
                            roomIndex, s));

                    jTableSensor.setModel(new ValueSourceAbstractTableModel(
                        valueSourceList));
                    jTableSensor.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
                    jTableSensor.getColumnModel().getColumn(0).setPreferredWidth(
                        JTABLE_SENSOR_COL0_WIDTH);
                    jTableSensor.getColumnModel().getColumn(1).setPreferredWidth(
                        JTABLE_SENSOR_COL1_WIDTH);

                }

            }
        } catch(NullPointerException ex) {
            System.out.println("-->" + ex.toString());
        }

    }//GEN-LAST:event_jButtonAddSensorToListActionPerformed

    private void jButtonRemoveSensorFromListActionPerformed(java.awt.event.
        ActionEvent evt) {//GEN-FIRST:
        event_jButtonRemoveSensorFromListActionPerformed

        if(jTableSensor.getSelectedRow() >= 0) {
            ValueSource source = ((ValueSourceAbstractTableModel)jTableSensor.
                getModel()).getElementAt(jTableSensor.getSelectedRow());

            ((BuildingTreeModel)jTreeBuilding.getModel()).removeSensor(source.
                floorIndex, source.roomIndex, source.sensorIndex);
            ((ValueSourceAbstractTableModel)jTableSensor.getModel()).
                removeValueSource(jTableSensor.getSelectedRow());
        }
        jButtonRemoveSensorFromList.setEnabled(false);jTableSensor.
            setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        jTableSensor.getColumnModel().getColumn(0).setPreferredWidth(
            JTABLE_SENSOR_COL0_WIDTH);
        jTableSensor.getColumnModel().getColumn(1).setPreferredWidth(
            JTABLE_SENSOR_COL1_WIDTH);

    }//GEN-LAST:event_jButtonRemoveSensorFromListActionPerformed
```

```
    private void jButtonRemoveFromTreeActionPerformed(java.awt.event.ActionEvent
        evt) {//GEN-FIRST:event_jButtonRemoveFromTreeActionPerformed

        try {

            if(jTreeBuilding.getLastSelectedPathComponent().getClass() == Building.
                class) {

                JOptionPane.showMessageDialog(this,
                    "Can not remove Building.",
                    "Error",
                    JOptionPane.ERROR_MESSAGE);

            } else if(jTreeBuilding.getLastSelectedPathComponent().getClass() ==
                Floor.class) {

                int floorIndex = (((BuildingTreeModel)jTreeBuilding.getModel()).
                    getIndexOfChild(((BuildingTreeModel)jTreeBuilding.getModel()).
                    getBuilding(), jTreeBuilding.getSelectionPath().
                    getLastPathComponent()));

                ((BuildingTreeModel)jTreeBuilding.getModel()).removeFloor(
                    floorIndex);

            } else if(jTreeBuilding.getLastSelectedPathComponent().getClass() ==
                Room.class) {

                int floorIndex = (((BuildingTreeModel)jTreeBuilding.getModel()).
                    getIndexOfChild(((BuildingTreeModel)jTreeBuilding.getModel()).
                    getBuilding(), jTreeBuilding.getSelectionPath().getParentPath()
                    .getLastPathComponent()));
                int roomIndex = (((BuildingTreeModel)jTreeBuilding.getModel()).
                    getIndexOfChild(jTreeBuilding.getSelectionPath().getParentPath
                    ().getLastPathComponent(), jTreeBuilding.getSelectionPath().
                    getLastPathComponent()));

                ((BuildingTreeModel)jTreeBuilding.getModel()).removeRoom(floorIndex
                    , roomIndex);

            }

            jButtonRemoveFromTree.setEnabled(false);

        } catch(NullPointerException ex) {
            System.out.println("--> " + ex.toString());
        }

    }//GEN-LAST:event_jButtonRemoveFromTreeActionPerformed

    private void jTableRuleMouseClicked(java.awt.event.MouseEvent evt) {//GEN-FIRST
        :event_jTableRuleMouseClicked
        if(jTableRule.getSelectedRow() >= 0) {
            jButtonRemoveRuleFromList.setEnabled(true);
        } else {
            jButtonRemoveRuleFromList.setEnabled(false);
        }
    }//GEN-LAST:event_jTableRuleMouseClicked

    private void jTableSensorMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
        FIRST:event_jTableSensorMouseClicked
        if(jTableSensor.getSelectedRow() >= 0) {
            jButtonRemoveSensorFromList.setEnabled(true);
        } else {
            jButtonRemoveSensorFromList.setEnabled(false);
        }
    }//GEN-LAST:event_jTableSensorMouseClicked

    private void jButtonPlayPauseActionPerformed(java.awt.event.ActionEvent evt) {
        //GEN-FIRST:event_jButtonPlayPauseActionPerformed
```

```
        if(buttonPlay == true) {
            buttonPlay = false;
            jButtonPlayPause.setIcon(new javax.swing.ImageIcon(getClass().
                getResource("/be/woine/thesis/utilz/Pause-Pressed-icon.png")));

            monitorAgent = new MonitorAgent("monitor_agent", jTextAreaMonitoring);
            monitorAgent.start();
        } else {
            buttonPlay = true;
            jButtonPlayPause.setIcon(new javax.swing.ImageIcon(getClass().
                getResource("/be/woine/thesis/utilz/Play-Normal-icon.png")));

            monitorAgent.stopAgent();
        }

    }//GEN-LAST:event_jButtonPlayPauseActionPerformed

    private void jTabbedPaneStateChanged(javax.swing.event.ChangeEvent evt) {//GEN-
        FIRST:event_jTabbedPaneStateChanged

        if(jTabbedPane.getSelectedIndex() == 1) {

            int samplingTime = (Integer.valueOf(jSpinnerSamplingPeriod.getValue().
                toString()))*1000;
            ArrayList<Reaction> reactionList = new ArrayList<Reaction>();

            // Add monitor and lease time on all tuple centres composed by '
                floor_room'
            ArrayList<String> tcList = getTupleCentreListFromBuilding(((
                BuildingTreeModel)jTreeBuilding.getModel()).getBuilding());
            for(int tc = 0; tc < tcList.size(); tc++) {
                reactionList.add(UniversalParser.getMonitorReaction(tcList.get(tc))
                    );
                reactionList.addAll(UniversalParser.getLeaseTimeReaction(tcList.get
                    (tc), samplingTime));
            }

            // Add rule related reactions
            if(!(((RuleAbstractTableModel)jTableRule.getModel()).getRuleList().
                isEmpty())) {

                for(int i = 0; i < ((RuleAbstractTableModel)jTableRule.getModel()).
                    getRuleList().size(); i++) {

                    try {
                        reactionList.addAll(UniversalParser.getReactionFromRule(((
                            RuleAbstractTableModel)jTableRule.getModel()).
                            getRuleList().get(i), samplingTime));
                    } catch (UndefinedRuleException ex) {
                        System.err.println(ex.toString());
                    } catch (InvalidOperatorException ex) {
                        System.err.println(ex.toString());
                    }

                }
             }

            // Compute JTree and JTable
            if(!reactionList.isEmpty()) {

                ArrayList<TupleCentre> tupleCentreList = UniversalParser.
                    getTupleCentreListFromReaction(reactionList);
                NodeEnvironment nodeEnv = new NodeEnvironment(((BuildingTreeModel)
                    jTreeBuilding.getModel()).getRoot().toString(), tupleCentreList
                    );

                jTreeTupleCentre.setModel(new TupleCentreTreeModel(nodeEnv));
                jTableReaction.setModel(new ReactionAbstractTableModel());
                jTableReaction.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
```

```
                jTableReaction.getColumnModel().getColumn(0).setPreferredWidth(
                    JTABLE_REACTION_COL0_WIDTH);
                jTableReaction.getColumnModel().getColumn(1).setPreferredWidth(
                    JTABLE_REACTION_COL1_WIDTH);
                jTableReaction.getColumnModel().getColumn(2).setPreferredWidth(
                    JTABLE_REACTION_COL2_WIDTH);
                jTableReaction.getSelectionModel().setSelectionMode(
                    ListSelectionModel.SINGLE_SELECTION);
            }


        }

    }//GEN-LAST:event_jTabbedPaneStateChanged

    private void jButtonLaunchTucsonNodeActionPerformed(java.awt.event.ActionEvent
        evt) {//GEN-FIRST:event_jButtonLaunchTucsonNodeActionPerformed

        try {
            launchNode = new LaunchNode();
            launchNode.start();
            jButtonLaunchTucsonNode.setEnabled(false);
            jButtonUpdateRuleList.setEnabled(true);
            jTextFieldTucsonNodeStatus.setText("TuCSoN Node launched ...");

            rulerAgent = new RulerAgent("ruler_agent", ((RuleAbstractTableModel)
                jTableRule.getModel()).getRuleList());
            rulerAgent.start();

            buildingAgent = new BuildingAgent("building_agent", building);
            buildingAgent.start();

            nodeLaunched = true;

            try {
                ctx = new TucsonContextSynch(Tucson.getContext(new AgentId("
                    spec_agent")));
            } catch (ContextNotAvailableException ex) {
                System.err.println(ex);
            }

        } catch (TucsonException ex) {
            jTextFieldTucsonNodeStatus.setText("Can not launch TuCSoN Node : " + ex
                .getMessage());
        }

    }//GEN-LAST:event_jButtonLaunchTucsonNodeActionPerformed

    private void jButtonUpdateRuleListActionPerformed(java.awt.event.ActionEvent
        evt) {//GEN-FIRST:event_jButtonUpdateRuleListActionPerformed
        rulerAgent.updateRuleList(((RuleAbstractTableModel)jTableRule.getModel()).
            getRuleList());
    }//GEN-LAST:event_jButtonUpdateRuleListActionPerformed

    private void jButtonApplyToTupleCentreActionPerformed(java.awt.event.
        ActionEvent evt) {//GEN-FIRST:
        event_jButtonApplyToTupleCentreActionPerformed

        if(nodeLaunched) {
            String message = setReactionToTupleCentre(false);
            JOptionPane.showMessageDialog(this,
                message,
                "Specification",
                JOptionPane.INFORMATION_MESSAGE);
            ((TupleCentreTreeModel)jTreeTupleCentre.getModel()).reload();
        } else {
            JOptionPane.showMessageDialog(this,
                "TuCSoN Node is not yet launched.",
                "Error",
                JOptionPane.ERROR_MESSAGE);
```

```
        }

    }//GEN-LAST:event_jButtonApplyToTupleCentreActionPerformed

    private void jButtonApplyToAllTupleCentreActionPerformed(java.awt.event.
        ActionEvent evt) {//GEN-FIRST:
        event_jButtonApplyToAllTupleCentreActionPerformed

        if(nodeLaunched) {
            resetAllTupleCentres();
            String message = setReactionToTupleCentre(true);
            JOptionPane.showMessageDialog(this,
                message,
                "Specification",
                JOptionPane.INFORMATION_MESSAGE);
            ((TupleCentreTreeModel)jTreeTupleCentre.getModel()).reload();
        } else {
            JOptionPane.showMessageDialog(this,
                "TuCSoN Node is not yet launched.",
                "Error",
                JOptionPane.ERROR_MESSAGE);
        }

    }//GEN-LAST:event_jButtonApplyToAllTupleCentreActionPerformed

    private void jButtonResetTupleCentreActionPerformed(java.awt.event.ActionEvent
        evt) {//GEN-FIRST:event_jButtonResetTupleCentreActionPerformed

        if(nodeLaunched)
            resetAllTupleCentres();

    }//GEN-LAST:event_jButtonResetTupleCentreActionPerformed

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButtonAddRuleToList;
    private javax.swing.JButton jButtonAddSensorToList;
    private javax.swing.JButton jButtonAddToTree;
    private javax.swing.JButton jButtonApplyToAllTupleCentre;
    private javax.swing.JButton jButtonApplyToTupleCentre;
    private javax.swing.JButton jButtonEditRuleInList;
    private javax.swing.JButton jButtonEditSensorInList;
    private javax.swing.JButton jButtonLaunchTucsonNode;
    private javax.swing.JButton jButtonModifyInTree;
    private javax.swing.JButton jButtonPlayPause;
    private javax.swing.JButton jButtonRemoveFromTree;
    private javax.swing.JButton jButtonRemoveRuleFromList;
    private javax.swing.JButton jButtonRemoveSensorFromList;
    private javax.swing.JButton jButtonResetTupleCentre;
    private javax.swing.JButton jButtonUpdateRuleList;
    private javax.swing.JLabel jLabelBuildingOfInterest;
    private javax.swing.JLabel jLabelMonitor;
    private javax.swing.JLabel jLabelReactions;
    private javax.swing.JLabel jLabelRuleList;
    private javax.swing.JLabel jLabelSamplingPeriod;
    private javax.swing.JLabel jLabelSensorList;
    private javax.swing.JLabel jLabelTucsonNodeStatus;
    private javax.swing.JLabel jLabelTupleCentres;
    private javax.swing.JLabel jLabelUnit;
    private javax.swing.JMenuBar jMenuBar;
    private javax.swing.JMenu jMenuEdit;
    private javax.swing.JMenu jMenuFile;
    private javax.swing.JPanel jPanelTabDeveloper;
    private javax.swing.JPanel jPanelTabUser;
    private javax.swing.JPanel jPanelTucsonNode;
    private javax.swing.JScrollPane jScrollPaneMonitoring;
    private javax.swing.JScrollPane jScrollPaneReactions;
    private javax.swing.JScrollPane jScrollPaneRuleTable;
    private javax.swing.JScrollPane jScrollPaneSensorTable;
    private javax.swing.JScrollPane jScrollPaneTree;
    private javax.swing.JScrollPane jScrollPaneTupleCentre;
```

```
    private javax.swing.JSpinner jSpinnerSamplingPeriod;
    private javax.swing.JTabbedPane jTabbedPane;
    private javax.swing.JTable jTableReaction;
    private javax.swing.JTable jTableRule;
    private javax.swing.JTable jTableSensor;
    private javax.swing.JTextArea jTextAreaMonitoring;
    private javax.swing.JTextField jTextFieldTucsonNodeStatus;
    private javax.swing.JTree jTreeBuilding;
    private javax.swing.JTree jTreeTupleCentre;
    // End of variables declaration//GEN-END:variables
}
```

Listing M.1 – EnvironmentManagerForm Class

Annexe N

package be.woine.thesis.actionManagerImpl

```
package be.woine.thesis.actionManagerImpl;

import be.woine.thesis.actionManagerGui.ActionManagerForm;

/**
 *
 * @author Remy
 * @version 1.0
 */
public class ActionManager {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        ActionManagerForm actionManagerForm = new ActionManagerForm();
        actionManagerForm.setVisible(true);

    }
}
```

Listing N.1 – ActionManager Class

Annexe O

package be.woine.thesis.actionManagerGui

```
/*
 * ActionManagerForm.java
 *
 * Created on 26-mai-2012, 9:59:36
 */
package be.woine.thesis.actionManagerGui;

import be.woine.thesis.agent.ActionAgent;
import java.awt.Toolkit;
import java.util.Locale;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import javax.swing.InputVerifier;
import javax.swing.JComponent;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

/**
 *
 * @author Remy
 */
public class ActionManagerForm extends javax.swing.JFrame {

    private ActionAgent actionAgent;
    private String nodeAddress;

    /** Creates new form ActionManagerForm */
    public ActionManagerForm() {

        /* Look And Feel */
        try {
            javax.swing.UIManager.setLookAndFeel(javax.swing.UIManager.
                getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException ex) {
            System.out.println(ex.toString());
        } catch (InstantiationException ex) {
            System.out.println(ex.toString());
        } catch (IllegalAccessException ex) {
            System.out.println(ex.toString());
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            System.out.println(ex.toString());
        }

        initComponents();

    }
```

```
    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:
        initComponents
    private void initComponents() {

        jPanelActionManager = new javax.swing.JPanel();
        jScrollPaneLogWindow = new javax.swing.JScrollPane();
        jTextPaneLogWindow = new javax.swing.JTextPane();
        jLabelLogWindow = new javax.swing.JLabel();
        jLabelAgentName = new javax.swing.JLabel();
        jTextFieldAgentName = new javax.swing.JTextField();
        jButtonStartAgent = new javax.swing.JButton();
        jButtonExit = new javax.swing.JButton();
        jPanelActionPrintConsole = new javax.swing.JPanel();
        jScrollPaneConsole = new javax.swing.JScrollPane();
        jTextAreaConsole = new javax.swing.JTextArea();
        jPanelActionDialogAlert = new javax.swing.JPanel();
        jLabelNumberOfDialog = new javax.swing.JLabel();
        jTextFieldAlertDialogNumber = new javax.swing.JTextField();
        jLabelEnvironmentManagerIPAddress = new javax.swing.JLabel();
        jCheckBoxLocalhost = new javax.swing.JCheckBox();
        jTextFieldIPAddress = new javax.swing.JTextField();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Action␣Manager");
        setLocationByPlatform(true);
        setResizable(false);

        jPanelActionManager.setBorder(javax.swing.BorderFactory.createTitledBorder(
            null, "Action␣Manager", javax.swing.border.TitledBorder.
            DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.DEFAULT_POSITION
            , new java.awt.Font("Tahoma", 1, 11), new java.awt.Color(0, 0, 255)));
            // NOI18N

        jTextPaneLogWindow.setEditable(false);
        jScrollPaneLogWindow.setViewportView(jTextPaneLogWindow);

        jLabelLogWindow.setFont(new java.awt.Font("Tahoma", 1, 11));
        jLabelLogWindow.setText("Log␣Window␣:");

        jLabelAgentName.setFont(new java.awt.Font("Tahoma", 1, 11));
        jLabelAgentName.setText("Agent␣Name␣:");

        jTextFieldAgentName.setText("action_man");

        jButtonStartAgent.setText("Start␣Agent");
        jButtonStartAgent.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonStartAgentActionPerformed(evt);
            }
        });

        jButtonExit.setText("Exit");
        jButtonExit.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonExitActionPerformed(evt);
            }
        });

        jPanelActionPrintConsole.setBorder(javax.swing.BorderFactory.
            createTitledBorder(null, "Action␣:␣Print␣Console", javax.swing.border.
            TitledBorder.DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.
            DEFAULT_POSITION, new java.awt.Font("Tahoma", 1, 11), new java.awt.
            Color(0, 204, 51))); // NOI18N
```

```
        jTextAreaConsole.setColumns(20);
        jTextAreaConsole.setEditable(false);
        jTextAreaConsole.setFont(new java.awt.Font("Consolas", 0, 10));
        jTextAreaConsole.setRows(5);
        jScrollPaneConsole.setViewportView(jTextAreaConsole);

        javax.swing.GroupLayout jPanelActionPrintConsoleLayout = new javax.swing.
            GroupLayout(jPanelActionPrintConsole);
        jPanelActionPrintConsole.setLayout(jPanelActionPrintConsoleLayout);
        jPanelActionPrintConsoleLayout.setHorizontalGroup(
            jPanelActionPrintConsoleLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.LEADING)
            .addGroup(jPanelActionPrintConsoleLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jScrollPaneConsole, javax.swing.GroupLayout.
                    DEFAULT_SIZE, 333, Short.MAX_VALUE)
                .addContainerGap())
        );
        jPanelActionPrintConsoleLayout.setVerticalGroup(
            jPanelActionPrintConsoleLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                jPanelActionPrintConsoleLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jScrollPaneConsole, javax.swing.GroupLayout.
                    DEFAULT_SIZE, 112, Short.MAX_VALUE)
                .addContainerGap())
        );

        jPanelActionDialogAlert.setBorder(javax.swing.BorderFactory.
            createTitledBorder(null, "Action␣:␣Dialog␣Alert", javax.swing.border.
            TitledBorder.DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.
            DEFAULT_POSITION, new java.awt.Font("Tahoma", 1, 11), new java.awt.
            Color(0, 204, 51))); // NOI18N
        jPanelActionDialogAlert.setPreferredSize(new java.awt.Dimension(365, 161));

        jLabelNumberOfDialog.setText("Number␣of␣thrown␣dialogs␣:␣");

        jTextFieldAlertDialogNumber.setEditable(false);
        jTextFieldAlertDialogNumber.setHorizontalAlignment(javax.swing.JTextField.
            RIGHT);
        jTextFieldAlertDialogNumber.setText("0");

        javax.swing.GroupLayout jPanelActionDialogAlertLayout = new javax.swing.
            GroupLayout(jPanelActionDialogAlert);
        jPanelActionDialogAlert.setLayout(jPanelActionDialogAlertLayout);
        jPanelActionDialogAlertLayout.setHorizontalGroup(
            jPanelActionDialogAlertLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.LEADING)
            .addGroup(jPanelActionDialogAlertLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabelNumberOfDialog)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
                    )
                .addComponent(jTextFieldAlertDialogNumber, javax.swing.GroupLayout.
                    PREFERRED_SIZE, 42, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(164, Short.MAX_VALUE))
        );
        jPanelActionDialogAlertLayout.setVerticalGroup(
            jPanelActionDialogAlertLayout.createParallelGroup(javax.swing.
                GroupLayout.Alignment.LEADING)
            .addGroup(jPanelActionDialogAlertLayout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanelActionDialogAlertLayout.createParallelGroup(javax.
                    swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabelNumberOfDialog)
                    .addComponent(jTextFieldAlertDialogNumber, javax.swing.
                        GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.
                        DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(107, Short.MAX_VALUE))
```

```
        );

        jLabelEnvironmentManagerIPAddress.setText("Environment␣Manager␣IP␣Address␣:
            ");

        jCheckBoxLocalhost.setSelected(true);
        jCheckBoxLocalhost.setText("localhost");
        jCheckBoxLocalhost.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jCheckBoxLocalhostActionPerformed(evt);
            }
        });

        jTextFieldIPAddress.setEditable(false);
        jTextFieldIPAddress.setText("192.168.1.1");

        javax.swing.GroupLayout jPanelActionManagerLayout = new javax.swing.
            GroupLayout(jPanelActionManager);
        jPanelActionManager.setLayout(jPanelActionManagerLayout);
        jPanelActionManagerLayout.setHorizontalGroup(
            jPanelActionManagerLayout.createParallelGroup(javax.swing.GroupLayout.
                Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                jPanelActionManagerLayout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanelActionManagerLayout.createParallelGroup(javax.swing
                    .GroupLayout.Alignment.TRAILING)
                    .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
                        jPanelActionManagerLayout.createSequentialGroup()
                        .addComponent(jPanelActionPrintConsole, javax.swing.
                            GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.
                            DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(18, 18, 18)
                        .addComponent(jPanelActionDialogAlert, javax.swing.
                            GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.
                            DEFAULT_SIZE, Short.MAX_VALUE))
                    .addComponent(jScrollPaneLogWindow, javax.swing.GroupLayout.
                        Alignment.LEADING, javax.swing.GroupLayout.DEFAULT_SIZE,
                        748, Short.MAX_VALUE)
                    .addGroup(jPanelActionManagerLayout.createSequentialGroup()
                        .addGroup(jPanelActionManagerLayout.createParallelGroup(
                            javax.swing.GroupLayout.Alignment.LEADING)
                            .addGroup(jPanelActionManagerLayout.
                                createSequentialGroup()
                                .addGroup(jPanelActionManagerLayout.
                                    createParallelGroup(javax.swing.GroupLayout.
                                    Alignment.LEADING)
                                    .addComponent(jLabelAgentName)
                                    .addGroup(jPanelActionManagerLayout.
                                        createSequentialGroup()
                                        .addGap(10, 10, 10)
                                        .addComponent(jTextFieldAgentName, javax.
                                            swing.GroupLayout.PREFERRED_SIZE, 105,
                                            javax.swing.GroupLayout.PREFERRED_SIZE)
                                        ))
                                .addGap(70, 70, 70)
                                .addComponent(jLabelEnvironmentManagerIPAddress)
                                .addPreferredGap(javax.swing.LayoutStyle.
                                    ComponentPlacement.UNRELATED)
                                .addGroup(jPanelActionManagerLayout.
                                    createParallelGroup(javax.swing.GroupLayout.
                                    Alignment.LEADING)
                                    .addComponent(jTextFieldIPAddress, javax.swing.
                                        GroupLayout.PREFERRED_SIZE, 102, javax.
                                        swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jCheckBoxLocalhost)))
                            .addComponent(jLabelLogWindow))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                            .RELATED, 195, Short.MAX_VALUE)
```

```
                    .addGroup(jPanelActionManagerLayout.createParallelGroup(
                        javax.swing.GroupLayout.Alignment.TRAILING, false)
                        .addComponent(jButtonExit, javax.swing.GroupLayout.
                            DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                             Short.MAX_VALUE)
                        .addComponent(jButtonStartAgent))))
            .addContainerGap())
);
jPanelActionManagerLayout.setVerticalGroup(
    jPanelActionManagerLayout.createParallelGroup(javax.swing.GroupLayout.
        Alignment.LEADING)
    .addGroup(jPanelActionManagerLayout.createSequentialGroup()
        .addGroup(jPanelActionManagerLayout.createParallelGroup(javax.swing
            .GroupLayout.Alignment.LEADING)
            .addGroup(jPanelActionManagerLayout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanelActionManagerLayout.createParallelGroup(
                    javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabelAgentName)
                    .addComponent(jLabelEnvironmentManagerIPAddress)
                    .addComponent(jCheckBoxLocalhost))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                    .RELATED)
                .addGroup(jPanelActionManagerLayout.createParallelGroup(
                    javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jTextFieldAgentName, javax.swing.
                        GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout
                        .DEFAULT_SIZE, javax.swing.GroupLayout.
                        PREFERRED_SIZE)
                    .addComponent(jTextFieldIPAddress, javax.swing.
                        GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout
                        .DEFAULT_SIZE, javax.swing.GroupLayout.
                        PREFERRED_SIZE))
                .addGap(18, 18, 18)
                .addComponent(jLabelLogWindow))
            .addGroup(jPanelActionManagerLayout.createSequentialGroup()
                .addGap(19, 19, 19)
                .addComponent(jButtonStartAgent)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                    .RELATED)
                .addComponent(jButtonExit)))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED
            )
        .addComponent(jScrollPaneLogWindow, javax.swing.GroupLayout.
            DEFAULT_SIZE, 267, Short.MAX_VALUE)
        .addGap(18, 18, 18)
        .addGroup(jPanelActionManagerLayout.createParallelGroup(javax.swing
            .GroupLayout.Alignment.LEADING)
            .addComponent(jPanelActionDialogAlert, javax.swing.GroupLayout.
                DEFAULT_SIZE, 165, Short.MAX_VALUE)
            .addComponent(jPanelActionPrintConsole, javax.swing.GroupLayout
                .PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE))
        .addContainerGap())
);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane
    ());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jPanelActionManager, javax.swing.GroupLayout.
            DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
            MAX_VALUE)
        .addContainerGap())
);
layout.setVerticalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelActionManager, javax.swing.GroupLayout.
                    DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.
                    MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jButtonStartAgentActionPerformed(java.awt.event.ActionEvent evt) {
        //GEN-FIRST:event_jButtonStartAgentActionPerformed

        if(jTextFieldAgentName.getText().isEmpty()) {

            JOptionPane.showMessageDialog(this,
                "No Agent name mentionned.",
                "Error",
                JOptionPane.ERROR_MESSAGE);

        } else {
            if(actionAgent == null) {

                if(jCheckBoxLocalhost.isSelected())
                    nodeAddress = "localhost";
                else
                    nodeAddress = "'" + jTextFieldIPAddress.getText() + "'";

                actionAgent = new ActionAgent(jTextFieldAgentName.getText().
                    toLowerCase(Locale.FRENCH), nodeAddress, jTextPaneLogWindow,
                    jTextAreaConsole, jTextFieldAlertDialogNumber);
                actionAgent.start();
            }
        }

    }//GEN-LAST:event_jButtonStartAgentActionPerformed

    private void jButtonExitActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
        FIRST:event_jButtonExitActionPerformed

        if(actionAgent != null) {
            if(actionAgent.isAlive())
                actionAgent.stopAgent();
        }

    }//GEN-LAST:event_jButtonExitActionPerformed

    private void jCheckBoxLocalhostActionPerformed(java.awt.event.ActionEvent evt)
        {//GEN-FIRST:event_jCheckBoxLocalhostActionPerformed

        if(jCheckBoxLocalhost.isSelected()) {
            jTextFieldIPAddress.setEditable(false);
        } else {
            jTextFieldIPAddress.setEditable(true);
        }



    }//GEN-LAST:event_jCheckBoxLocalhostActionPerformed

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButtonExit;
    private javax.swing.JButton jButtonStartAgent;
    private javax.swing.JCheckBox jCheckBoxLocalhost;
    private javax.swing.JLabel jLabelAgentName;
    private javax.swing.JLabel jLabelEnvironmentManagerIPAddress;
    private javax.swing.JLabel jLabelLogWindow;
    private javax.swing.JLabel jLabelNumberOfDialog;
    private javax.swing.JPanel jPanelActionDialogAlert;
```

```
    private javax.swing.JPanel jPanelActionManager;
    private javax.swing.JPanel jPanelActionPrintConsole;
    private javax.swing.JScrollPane jScrollPaneConsole;
    private javax.swing.JScrollPane jScrollPaneLogWindow;
    private javax.swing.JTextArea jTextAreaConsole;
    private javax.swing.JTextField jTextFieldAgentName;
    private javax.swing.JTextField jTextFieldAlertDialogNumber;
    private javax.swing.JTextField jTextFieldIPAddress;
    private javax.swing.JTextPane jTextPaneLogWindow;
    // End of variables declaration//GEN-END:variables
}
```

Listing O.1 – ActionManagerForm Class

Annexe P

package be.woine.thesis.sensorEmulatorImpl

```
package be.woine.thesis.sensorEmulatorImpl;

import be.woine.thesis.sensorEmulatorGui.SensorEmulatorForm;

/**
 *
 * @author Remy
 * @version 1.0
 */
public class SensorEmulator {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        SensorEmulatorForm sensorEmulatorForm = new SensorEmulatorForm();
        sensorEmulatorForm.setVisible(true);

    }
}
```

Listing P.1 – SensorEmulator Class

Annexe Q

package be.woine.thesis.sensorEmulatorGui

```
/*
 * SensorEmulatorForm.java
 *
 * Created on 27-mai-2012, 16:17:46
 */
package be.woine.thesis.sensorEmulatorGui;

import alice.logictuple.InvalidTupleOperationException;
import alice.logictuple.LogicTuple;
import alice.logictuple.TupleArgument;
import alice.logictuple.Var;
import alice.tucson.api.AgentId;
import alice.tucson.api.ContextNotAvailableException;
import alice.tucson.api.InvalidTupleCentreIdException;
import alice.tucson.api.OperationNotAllowedException;
import alice.tucson.api.Tucson;
import alice.tucson.api.TucsonContextSynch;
import alice.tucson.api.TupleCentreId;
import alice.tucson.api.UnreachableNodeException;
import be.woine.thesis.building.common.Building;
import be.woine.thesis.building.common.Floor;
import be.woine.thesis.building.common.Room;
import be.woine.thesis.dialog.JDialogEmulateSensor;
import be.woine.thesis.model.BuildingTreeModel;
import be.woine.thesis.model.ValueSourceAbstractTableModel;
import be.woine.thesis.rule.common.ValueSource;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.util.ArrayList;
import javax.swing.JOptionPane;
import javax.swing.ListSelectionModel;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import javax.swing.tree.TreeSelectionModel;

/**
 *
 * @author Remy
 */
public class SensorEmulatorForm extends javax.swing.JFrame {

    private Building building;
    private ValueSource source;

    private TucsonContextSynch ctx;

    /** Creates new form SensorEmulatorForm */
```

```
public SensorEmulatorForm() {

    /* Look And Feel */
    try {
        javax.swing.UIManager.setLookAndFeel(javax.swing.UIManager.
            getSystemLookAndFeelClassName());
    } catch (ClassNotFoundException ex) {
        System.out.println(ex.toString());
    } catch (InstantiationException ex) {
        System.out.println(ex.toString());
    } catch (IllegalAccessException ex) {
        System.out.println(ex.toString());
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
        System.out.println(ex.toString());
    }

    initComponents();

    jTableSensor.getSelectionModel().setSelectionMode(ListSelectionModel.
        SINGLE_SELECTION);

    jTreeBuilding.getSelectionModel().setSelectionMode(TreeSelectionModel.
        SINGLE_TREE_SELECTION);
    jTreeBuilding.getSelectionModel().addTreeSelectionListener(new
        TreeSelectionListener() {

        @Override
        public void valueChanged(TreeSelectionEvent e) {

            ArrayList<ValueSource> sensorList = new ArrayList<ValueSource>();

            try {
                if(jTreeBuilding.getLastSelectedPathComponent().getClass() ==
                    Room.class) {

                    for(int f = 0; f < building.getFloorCount(); f++) {
                        for(int r = 0; r < building.getFloor(f).getRoomCount();
                             r++) {
                            if(building.getFloor(f).getRoom(r) == e.getPath().
                                getLastPathComponent()) {
                                for(int s = 0; s < building.getFloor(f).getRoom
                                    (r).getSensorCount(); s++) {
                                    sensorList.add(new ValueSource(building, f,
                                         r, s));
                                }
                            }
                        }
                    }

                } else if(jTreeBuilding.getLastSelectedPathComponent().getClass
                    () == Floor.class) {

                    for(int f = 0; f < building.getFloorCount(); f++) {
                        if(building.getFloor(f) == e.getPath().
                            getLastPathComponent()) {
                            for(int r = 0; r < building.getFloor(f).
                                getRoomCount(); r++) {
                                for(int s = 0; s < building.getFloor(f).getRoom
                                    (r).getSensorCount(); s++)
                                    sensorList.add(new ValueSource(building, f,
                                         r, s));
                            }
                        }
                    }

                } else if(jTreeBuilding.getLastSelectedPathComponent().getClass
                    () == Building.class) {

                    for(int f = 0; f < building.getFloorCount(); f++) {
```

```
                        for(int r = 0; r < building.getFloor(f).getRoomCount();
                             r++) {
                            for(int s = 0; s < building.getFloor(f).getRoom(r).
                                getSensorCount(); s++)
                                sensorList.add(new ValueSource(building, f, r,
                                    s));
                        }
                    }

                }

                jTableSensor.setModel(new ValueSourceAbstractTableModel(
                    sensorList));

            } catch(NullPointerException ex) {
                System.out.println(e.getSource() + " ---> " + ex.toString());
            }
        }
    });

}

/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:
    initComponents
private void initComponents() {

    jPanelSensorSelector = new javax.swing.JPanel();
    jScrollPaneTree = new javax.swing.JScrollPane();
    jTreeBuilding = new javax.swing.JTree();
    jScrollPaneSensorTable = new javax.swing.JScrollPane();
    jTableSensor = new javax.swing.JTable();
    jLabelBuilding = new javax.swing.JLabel();
    jLabelSensorList = new javax.swing.JLabel();
    jLabelSelectedSensor = new javax.swing.JLabel();
    jTextFieldSelectedSensor = new javax.swing.JTextField();
    jButtonEmulate = new javax.swing.JButton();
    jPanelTucsonNode = new javax.swing.JPanel();
    jButtonRetreiveBuilding = new javax.swing.JButton();
    jTextFieldBuildingStatus = new javax.swing.JTextField();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    jPanelSensorSelector.setBorder(javax.swing.BorderFactory.createTitledBorder
        (null, "Sensor Selector", javax.swing.border.TitledBorder.
        DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.DEFAULT_POSITION
        , new java.awt.Font("Tahoma", 1, 11), new java.awt.Color(0, 0, 255)));
        // NOI18N

    jTreeBuilding.setModel(new BuildingTreeModel(building));
    jScrollPaneTree.setViewportView(jTreeBuilding);

    jTableSensor.setModel(new ValueSourceAbstractTableModel());
    jTableSensor.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseClicked(java.awt.event.MouseEvent evt) {
            jTableSensorMouseClicked(evt);
        }
    });
    jScrollPaneSensorTable.setViewportView(jTableSensor);

    jLabelBuilding.setFont(new java.awt.Font("Tahoma", 1, 11));
    jLabelBuilding.setText("Building :");

    jLabelSensorList.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
    jLabelSensorList.setText("Sensors' List :");
```

```
jLabelSelectedSensor.setText("Selected Sensor :");

jTextFieldSelectedSensor.setEditable(false);

jButtonEmulate.setText("Emulate");
jButtonEmulate.setEnabled(false);
jButtonEmulate.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonEmulateActionPerformed(evt);
    }
});

jPanelTucsonNode.setBorder(javax.swing.BorderFactory.createTitledBorder(
    null, "TuCSoN Node", javax.swing.border.TitledBorder.
    DEFAULT_JUSTIFICATION, javax.swing.border.TitledBorder.DEFAULT_POSITION
    , new java.awt.Font("Tahoma", 1, 11), new java.awt.Color(0, 204, 51)));
     // NOI18N

jButtonRetreiveBuilding.setText("Retreive Building Object");
jButtonRetreiveBuilding.addActionListener(new java.awt.event.ActionListener
    () {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonRetreiveBuildingActionPerformed(evt);
    }
});

jTextFieldBuildingStatus.setEditable(false);

javax.swing.GroupLayout jPanelTucsonNodeLayout = new javax.swing.
    GroupLayout(jPanelTucsonNode);
jPanelTucsonNode.setLayout(jPanelTucsonNodeLayout);
jPanelTucsonNodeLayout.setHorizontalGroup(
    jPanelTucsonNodeLayout.createParallelGroup(javax.swing.GroupLayout.
        Alignment.LEADING)
    .addGroup(jPanelTucsonNodeLayout.createSequentialGroup()
        .addContainerGap()
        .addComponent(jButtonRetreiveBuilding, javax.swing.GroupLayout.
            PREFERRED_SIZE, 158, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.
            UNRELATED)
        .addComponent(jTextFieldBuildingStatus, javax.swing.GroupLayout.
            DEFAULT_SIZE, 299, Short.MAX_VALUE)
        .addContainerGap())
);
jPanelTucsonNodeLayout.setVerticalGroup(
    jPanelTucsonNodeLayout.createParallelGroup(javax.swing.GroupLayout.
        Alignment.LEADING)
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
        jPanelTucsonNodeLayout.createSequentialGroup()
        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.
            MAX_VALUE)
        .addGroup(jPanelTucsonNodeLayout.createParallelGroup(javax.swing.
            GroupLayout.Alignment.BASELINE)
            .addComponent(jButtonRetreiveBuilding)
            .addComponent(jTextFieldBuildingStatus, javax.swing.GroupLayout
                .PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(42, 42, 42))
);

javax.swing.GroupLayout jPanelSensorSelectorLayout = new javax.swing.
    GroupLayout(jPanelSensorSelector);
jPanelSensorSelector.setLayout(jPanelSensorSelectorLayout);
jPanelSensorSelectorLayout.setHorizontalGroup(
    jPanelSensorSelectorLayout.createParallelGroup(javax.swing.GroupLayout.
        Alignment.LEADING)
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
        jPanelSensorSelectorLayout.createSequentialGroup()
        .addContainerGap()
```

```
            .addGroup(jPanelSensorSelectorLayout.createParallelGroup(javax.
                swing.GroupLayout.Alignment.TRAILING)
                .addComponent(jPanelTucsonNode, javax.swing.GroupLayout.
                    Alignment.LEADING, javax.swing.GroupLayout.PREFERRED_SIZE,
                    javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.
                    GroupLayout.PREFERRED_SIZE)
                .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
                    jPanelSensorSelectorLayout.createSequentialGroup()
                    .addGroup(jPanelSensorSelectorLayout.createParallelGroup(
                        javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jScrollPaneTree, javax.swing.GroupLayout.
                            PREFERRED_SIZE, 158, javax.swing.GroupLayout.
                            PREFERRED_SIZE)
                        .addComponent(jLabelBuilding))
                    .addGap(18, 18, 18)
                    .addGroup(jPanelSensorSelectorLayout.createParallelGroup(
                        javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jLabelSensorList)
                        .addGroup(jPanelSensorSelectorLayout.
                            createSequentialGroup()
                            .addComponent(jScrollPaneSensorTable, javax.swing.
                                GroupLayout.PREFERRED_SIZE, 153, javax.swing.
                                GroupLayout.PREFERRED_SIZE)
                            .addGap(18, 18, 18)
                            .addGroup(jPanelSensorSelectorLayout.
                                createParallelGroup(javax.swing.GroupLayout.
                                Alignment.TRAILING)
                                .addComponent(jLabelSelectedSensor, javax.swing
                                    .GroupLayout.Alignment.LEADING)
                                .addComponent(jTextFieldSelectedSensor, javax.
                                    swing.GroupLayout.Alignment.LEADING, javax.
                                    swing.GroupLayout.DEFAULT_SIZE, 152, Short.
                                    MAX_VALUE)
                                .addComponent(jButtonEmulate))))))
            .addContainerGap())
        );
        jPanelSensorSelectorLayout.setVerticalGroup(
            jPanelSensorSelectorLayout.createParallelGroup(javax.swing.GroupLayout.
                Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
                jPanelSensorSelectorLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelTucsonNode, javax.swing.GroupLayout.
                    PREFERRED_SIZE, 81, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)
                .addGroup(jPanelSensorSelectorLayout.createParallelGroup(javax.
                    swing.GroupLayout.Alignment.LEADING)
                    .addGroup(jPanelSensorSelectorLayout.createSequentialGroup()
                        .addGroup(jPanelSensorSelectorLayout.createParallelGroup(
                            javax.swing.GroupLayout.Alignment.BASELINE)
                            .addComponent(jLabelBuilding)
                            .addComponent(jLabelSensorList))
                        .addGap(76, 76, 76)
                        .addComponent(jLabelSelectedSensor)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                            .RELATED)
                        .addComponent(jTextFieldSelectedSensor, javax.swing.
                            GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.
                            DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement
                            .UNRELATED)
                        .addComponent(jButtonEmulate))
                    .addGroup(jPanelSensorSelectorLayout.createSequentialGroup()
                        .addGap(20, 20, 20)
                        .addGroup(jPanelSensorSelectorLayout.createParallelGroup(
                            javax.swing.GroupLayout.Alignment.LEADING, false)
                            .addComponent(jScrollPaneTree, 0, 0, Short.MAX_VALUE)
                            .addComponent(jScrollPaneSensorTable, javax.swing.
                                GroupLayout.PREFERRED_SIZE, 242, javax.swing.
                                GroupLayout.PREFERRED_SIZE)))))
```

```
                .addGap(11, 11, 11))
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane
            ());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 551, Short.MAX_VALUE)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelSensorSelector, javax.swing.GroupLayout.
                    PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.
                    swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 432, Short.MAX_VALUE)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanelSensorSelector, javax.swing.GroupLayout.
                    PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.
                    swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jTableSensorMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
        FIRST:event_jTableSensorMouseClicked
        String str = "";
        source = ((ValueSourceAbstractTableModel)jTableSensor.getModel()).
            getElementAt(jTableSensor.getSelectedRow());

        str += jTableSensor.getModel().getValueAt(jTableSensor.getSelectedRow(), 0)
            .toString();
        str += "@{";
        str += source.building.getFloor(source.floorIndex).getName();
        str += "_";
        str += source.building.getFloor(source.floorIndex).getRoom(source.roomIndex
            ).getName();
        str += "}";

        jTextFieldSelectedSensor.setText(str);
    }//GEN-LAST:event_jTableSensorMouseClicked

    private void jButtonRetreiveBuildingActionPerformed(java.awt.event.ActionEvent
        evt) {//GEN-FIRST:event_jButtonRetreiveBuildingActionPerformed

        try {

            ctx = new TucsonContextSynch(Tucson.getContext(new AgentId("
                emulator_master")));
            TupleCentreId tcid = new TupleCentreId("exchange_centre");

            ctx.out(tcid, new LogicTuple("building_req"));

            LogicTuple template = new LogicTuple("building", new Var());

            LogicTuple req = ctx.in(tcid, template);

            TupleArgument[] rtup = req.getArg(0).toArray();
            byte[] objectToGet = new byte[rtup.length];

            for(int j = 0; j < rtup.length; j++) {
                objectToGet[j] = (byte)rtup[j].intValue();
            }
```

```
            // Input Stream
            ByteArrayInputStream bais = new ByteArrayInputStream(objectToGet);
            ObjectInputStream ois = new ObjectInputStream(bais);
            ois.close();

            // Read the object
            this.building = (Building)ois.readObject();
            jTreeBuilding.setModel(new BuildingTreeModel(this.building));

            jTextFieldBuildingStatus.setText("Building '" + this.building.getName()
                + "' retreived.");
            jButtonEmulate.setEnabled(true);
            jButtonRetreiveBuilding.setEnabled(false);

        } catch (ContextNotAvailableException ex) {
            System.err.println(ex);
            jTextFieldBuildingStatus.setText(ex.toString());
        } catch (InvalidTupleCentreIdException ex) {
            System.err.println(ex);
            jTextFieldBuildingStatus.setText(ex.toString());
        } catch (OperationNotAllowedException ex) {
            System.err.println(ex);
            jTextFieldBuildingStatus.setText(ex.toString());
        } catch (UnreachableNodeException ex) {
            System.err.println(ex);
            jTextFieldBuildingStatus.setText(ex.toString());
        } catch (InvalidTupleOperationException ex) {
            System.err.println(ex);
            jTextFieldBuildingStatus.setText(ex.toString());
        } catch (ClassNotFoundException ex) {
            System.err.println(ex);
            jTextFieldBuildingStatus.setText(ex.toString());
        } catch (IOException ex) {
            System.err.println(ex);
            jTextFieldBuildingStatus.setText(ex.toString());
        }

        if(ctx != null) {
            try {
                ctx.exit();
            } catch (OperationNotAllowedException ex) {
                System.err.println(ex);
            jTextFieldBuildingStatus.setText(ex.toString());
            }
        }
    }//GEN-LAST:event_jButtonRetreiveBuildingActionPerformed

    private void jButtonEmulateActionPerformed(java.awt.event.ActionEvent evt) {//
        GEN-FIRST:event_jButtonEmulateActionPerformed

        if((jTableSensor.getSelectedRow() != -1) && (!jTextFieldSelectedSensor.
            getText().isEmpty())) {
            JDialogEmulateSensor dialog = new JDialogEmulateSensor(((
                ValueSourceAbstractTableModel)jTableSensor.getModel()).getElementAt
                (jTableSensor.getSelectedRow()), null, false);
            dialog.setLocationByPlatform(true);
            dialog.setVisible(true);
        } else {
            JOptionPane.showMessageDialog(this,
                "No sensor selected.",
                "Error",
                JOptionPane.ERROR_MESSAGE);
        }

    }//GEN-LAST:event_jButtonEmulateActionPerformed


    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButtonEmulate;
    private javax.swing.JButton jButtonRetreiveBuilding;
```

```
    private javax.swing.JLabel jLabelBuilding;
    private javax.swing.JLabel jLabelSelectedSensor;
    private javax.swing.JLabel jLabelSensorList;
    private javax.swing.JPanel jPanelSensorSelector;
    private javax.swing.JPanel jPanelTucsonNode;
    private javax.swing.JScrollPane jScrollPaneSensorTable;
    private javax.swing.JScrollPane jScrollPaneTree;
    private javax.swing.JTable jTableSensor;
    private javax.swing.JTextField jTextFieldBuildingStatus;
    private javax.swing.JTextField jTextFieldSelectedSensor;
    private javax.swing.JTree jTreeBuilding;
    // End of variables declaration//GEN-END:variables
}
```

Listing Q.1 – SensorEmulatorForm Class